Broadview®
www.broadview.com.cn

神经网络与深度学习

吴岸城 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

作者简介

吴岸城

致力于深度学习在文本、图像领域的应用。曾在中兴通讯、亚信联创担任研发经理、技术经理等职务，现任菱歌科技首席算法科学家一职。

神经网络与深度学习

吴岸城 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

这是一本介绍神经网络和深度学习算法基本原理及相关实例的书籍，它不是教科书，作者已尽量把公式减少到最少，以适应绝大部分人的阅读基础和知识储备。本书涵盖了神经网络的研究历史、基础原理、深度学习中的自编码器、深度信念网络、卷积神经网络等，这些算法都已在很多行业发挥了价值。

本书适合有志于从事深度学习行业的，或想了解深度学习到底是什么的，或是有一定机器学习基础的朋友阅读。

图书在版编目（CIP）数据

神经网络与深度学习 / 吴岸城著. —北京：电子工业出版社，2016.6
ISBN 978-7-121-28869-2

Ⅰ. ①神… Ⅱ. ①吴… Ⅲ. ①人工神经网络②人工智能 Ⅳ. ①TP18

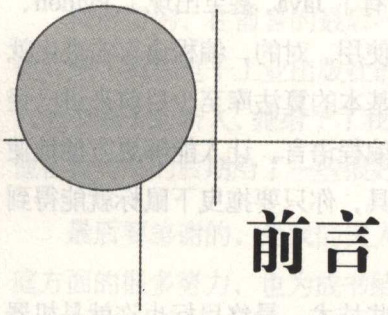
中国版本图书馆 CIP 数据核字（2016）第 109737 号

策划编辑：刘　皎
责任编辑：郑柳洁
印　　刷：北京季蜂印刷有限公司
装　　订：北京季蜂印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：720×1000　1/16　　印张：14.5　　字数：232 千字
版　　次：2016 年 6 月第 1 版
印　　次：2016 年 10 月第 4 次印刷
定　　价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。



前言

很多朋友告诉我，一本书总是要加一个前言才算完整。如果书没有前言，就好像只有山没有水一样，没有意境。

对我来说，这是我的第一本技术科普类读物。之所以把它称作第一本，是因为我从前没写过书，哪怕是一篇超过 4 万字的文章（论文不算）都没写过，所以听编辑说写书有字数要求时，我都没有概念，心想不就写本书吗？easy!

写着写着发现不对了，自己没有为一本书建立好整体知识体系！从 2014 年开始断断续续地写着，中间有段时间甚至想过放弃。我之所以没放弃，无非是因为觉得做事要有始有终。如果我写得不好，那是我的能力有限；如果因为一些之前估计不到的难度就放弃了，那是态度问题！

为什么说这是一本科普类读物呢？至少在我写书时，很多人（都是 IT、软件这个行业的人）对于神经网络、深度学习（Deep Learning）等都毫无概念，如果连这些人对神经网络等都没有概念，可以想象其普及程度有多低。但我觉得深度学习并不是只有大学学府或几个相关的专业学生才能研究它；并不是只有公司里这个领域的专家才能研究它，它是属于整个大众的东西。

对于技术层面的东西，将会慢慢简化再简化，如同编程语言一样，开始是汇编语言，后来是 C 语言，再后来有了 C++，再后来有了 Java，甚至出现了 Python、JavaScript，它们降低了进入门槛，可以让更多人使用。对的，编程语言的进化就是让更多人更便捷地使用。对于深度学习来说，基本的算法库至少目前来讲已经很多很多了，这些算法库基本覆盖了我们的现代编程语言，让人能够更方便地使用。微软甚至出了一个图形化的深度学习在线工具，你只要拖曳下鼠标就能得到一个算法并训练它，极大地加快了学习效率。

我强调这一点是想说：技术的进步扩散了这些技术，最终目标也许就是机器像人类那样思考，让人类想什么有什么，而不仅仅局限于技术层面；而今天深度学习的进化已经可以使机器通过学习已有的知识就能推导出或预测出未知的事物，想起这点时常让我激动，让我觉得创造出一个机器生命体是有可能的！所以写本书的意义在于让人们不过多地关注公式及推导过程，而是关注它的使用方法，把人类的想法迅速转换成生产力才是目的，毕竟只有人类的想法才是最有价值的！

按以上思路，我安排书的整体目录架构如下。

第 0 章，介绍机器学习、神经网络的历史，好让大家有基本的了解。

第 1 章，解释大脑的运作结构和如何利用仿生学产生逻辑上的神经元和神经网络。

第 2 章，我们用仿生学的知识试着构造一个神经网络（感知机）并使用它做些事情，解释了 XOR 问题。在 2.6 节给出一些例子，让我们能更好地了解神经网络是如何分类学习和预测的。

第 3 章，介绍深度学习的基本概念，深度学习和神经网络的联系。

第 4 章，介绍深度学习的常用方法。

第 5 章，介绍 AlphaGo。

第 6 章，两个重要概念，迁移学习和概率图模型 PGM。

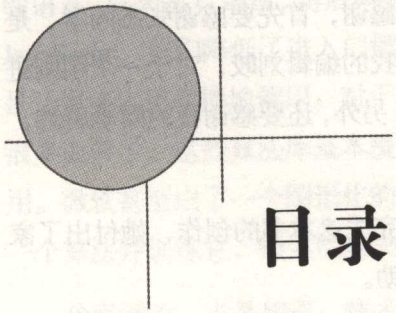
第7章，给出了一些经验以加快大家学习和研究的效率。

按照惯例，在前言的最后一部分应该做一些感谢，首先要感谢张杰同学，是他介绍了我和电子工业出版社的编辑认识。感谢我的编辑刘皎，对于一个刚刚进入写书圈子的新人，她给予了我很多帮助和支持。另外，还要感谢我的同事唐炜，他在我写书的后期给了一些很好的建议。

最后要感谢的，是我的夫人李茉，为了让我完成这本书的创作，她付出了家庭方面的很多努力，也为成书给了很多建议和帮助。

谢谢他们!

本书部分资料来源于网上，由于链接失效或无法知道原作者，因此没办法注明来源。请原作者看到后和我联系：wuanch@gmail.com。



目录

第 0 章　写在前面：神经网络的历史 1

第 1 章　神经网络是个什么东西 13

1.1　买橙子和机器学习 13

1.1.1　规则列表 14

1.1.2　机器学习 15

1.2　怎么定义神经网络 16

1.3　先来看看大脑如何学习 16

1.3.1　信息输入 17

1.3.2　模式加工 17

1.3.3　动作输出 18

1.4　生物意义上的神经元 19

1.4.1　神经元是如何工作的 19

1.4.2　组成神经网络 22

1.5　大脑如何解决现实生活中的分类问题 24

第 2 章　构造神经网络 26

2.1　构造一个神经元 26

2.2　感知机 30

2.3 感知机的学习 32
2.4 用代码实现一个感知机 34
2.4.1 Neuroph：一个基于 Java 的神经网络框架 34
2.4.2 代码实现感知机 37
2.4.3 感知机学习一个简单逻辑运算 39
2.4.4 XOR 问题 42
2.5 构造一个神经网络 44
2.5.1 线性不可分 45
2.5.2 解决 XOR 问题（解决线性不可分） 49
2.5.3 XOR 问题的代码实现 51
2.6 解决一些实际问题 54
2.6.1 识别动物 54
2.6.2 我是预测大师 59
第 3 章 深度学习是个什么东西 66
3.1 机器学习 67
3.2 特征 75
3.2.1 特征粒度 75
3.2.2 提取浅层特征 76
3.2.3 结构性特征 78
3.3 浅层学习和深度学习 81
3.4 深度学习和神经网络 83
3.5 如何训练神经网络 84
3.5.1 BP 算法：神经网络训练 84
3.5.2 BP 算法的问题 85
3.6 总结深度学习及训练过程 86
第 4 章 深度学习的常用方法 89
4.1 模拟大脑的学习和重构 90
4.1.1 灰度图像 91
4.1.2 流行感冒 92

4.1.3 看看如何编解码 93
4.1.4 如何训练 95
4.1.5 有监督微调 97
4.2 快速感知：稀疏编码（Sparse Coding） 98
4.3 栈式自编码器 100
4.4 解决概率分布问题：限制波尔兹曼机 102
4.4.1 生成模型和概率模型 102
4.4.2 能量模型 107
4.4.3 RBM 的基本概念 109
4.4.4 再看流行感冒的例子 111
4.5 DBN 112
4.6 卷积神经网络 114
4.6.1 卷积神经网络的结构 116
4.6.2 关于参数减少与权值共享 120
4.6.3 举个典型的例子：图片内容识别 124
4.7 不会忘记你：循环神经网络 131
4.7.1 什么是 RNN 131
4.7.2 LSTM 网络 136
4.7.3 LSTM 变体 141
4.7.4 结论 143
4.8 你是我的眼：利用稀疏编码器找图像的基本单位 143
4.9 你是我的眼（续） 150
4.10 使用深度信念网搞定花分类 160

第 5 章 深度学习的胜利：AlphaGo 169

5.1 AI 如何玩棋类游戏 169
5.2 围棋的复杂性 171
5.3 AlphaGo 的主要原理 173
5.3.1 策略网络 174
5.3.2 MCTS 拯救了围棋算法 176

5.3.3 强化学习：“周伯通，左右互搏” 179
5.3.4 估值网络 181
5.3.5 将所有组合到一起：树搜索 182
5.3.6 AlphaGo 有多好 185
5.3.7 总结 187
5.4 重要的技术进步 189
5.5 一些可以改进的地方 190
5.6 未来 192

第 6 章 两个重要的概念 194

6.1 迁移学习 194
6.2 概率图模型 197
6.2.1 贝叶斯的网络结构 201
6.2.2 概率图分类 204
6.2.3 如何应用 PGM 208

第 7 章 杂项 210

7.1 如何为不同类型的问题选择模型 210
7.2 我们如何学习“深度学习” 211
7.3 如何理解机器学习和深度学习的差异 212
7.4 大规模学习（Large Scale Learning）和并行计算 214
7.5 如果喜欢应用领域，可以考虑以下几种应用 215
7.6 类脑：人工智能的终极目标 216

参考文献 218

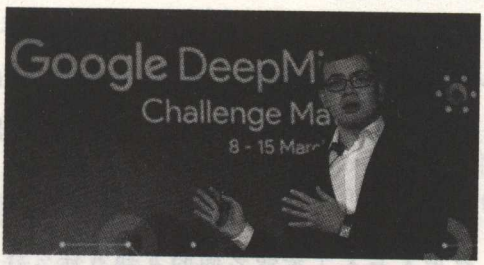
术语 220

0 写在前面：神经网络的历史

第一个片段

时间进入到 2016 年，在深度学习的历史中总绕不开这么几个片段，被人们津津乐道。

2016 年 3 月 10 日，在李世石败于 AlphaGo 的第一局对弈后，第二局对弈开始前 *The Verge* 对李世石和 DeepMind 创始人哈萨比斯（Hassabis）做了专访，不知道记者是不是故意的，当李世石和哈萨比斯同台时，我们看到了一个拘束且略带紧张的李世石，和一个比较放松，无所谓输赢心态的哈萨比斯。



DeepMind 创始人哈萨比斯

无论从哪种角度看，我都认为这是 AlphaGo 的一场自秀，虽说不在乎输赢，但也不确定赢的概率有多大，大家都有 50%的赢面。

整个访谈谈到围棋的部分较少，大部分话题都围绕着人工智能谈论。

记者们提出了以下几点好奇。

1. 哪些机器智能让你吃惊或者说哪些超出你的期望?

“是的。当 AlphaGo 穿越棋盘进入李世石占据优势的领地时，我们感到相当震惊，而且我认为从李世石的面部表情来看，他也很震惊。我认为那绝对是出乎所有人意料的一招。”

2. Google 对 AlphaGo 的帮助大吗？如果没有他们，你能完成这项工作吗?

Google 的帮助当然很大。AlphaGo 在硬件上的运行要求并不大，但是我们需要很多硬件设施去训练它、去测试不同的版本并且在 Google 的云端进行比赛训练。这些都对于硬件设施有很高的要求，因此我们不可能在没有这些资源的情况下，在这个时间范围内完成它。

然后，又提出了如下问题。

为了提升医疗保健水平，像 IBM 这样的公司已经开始了有关癌症诊断方面的工作，DeepMind 能带来什么呢?

让我们再聊聊智能手机助手吧。我发现你在当天的演讲中使用了电影《她》的一张图片，这难道就是最终的结果吗?

再让我们聊聊机器人吧。我驻扎在日本，这里被认为是机器人的精神家园。我认为机器人目前在这个国家在两个方向上被使用，Fanuc 这样的公司制造工业化机器人，他们能够在一个固定用途上做出令人惊讶的工作；而像软银的 Pepper 这样的公司则开发一些礼宾式的机器人，他们很有野心，却又用途有限。你的想法是什么?

目前你能看到的最直接使用学习方法的机器人案例是哪个?

你对未来人类、机器人与人工智能之间的互动有什么样的期待？很明显，人们目前都在科幻小说中幻想美丽的场景。

……

整个访谈都在谈人工智能在什么时间、什么方式给人类带来更多的助力。其实记者问的也是大多数普通人所想的，不关心纯粹的技术实现，更关心能给“我”带来什么好处。

DeepMind 创始人，现年 39 岁的哈萨比斯，在伦敦大学学院（UCL）念完了神经科学博士，发表了 12 篇研究论文，并继续在该校的盖茨比计算神经科学所（Gatsby Computational Neuroscience Unit）工作。

在 UCL 期间，哈萨比斯主要研究的是海马体。海马体是人脑的一个区域，对方向感、记忆调取和未来事件的想象至关重要。他与埃莉诺·马圭尔（Eleanor Maguire）教授密切合作，后者做出的一项发现是伦敦出租车司机的海马体比常人更大。

在 DeepMind 公司，哈萨比斯开始着手建立一套能够学习完成许多不同任务的通用算法。这套软件借鉴的发现之一是海马体将记忆牢牢印刻在人脑中，是通过在人睡眠期间高速播放这些记忆实现的。

在一场反映 DeepMind 技术灵活性的展示会中，该公司的程序只须原始像素输入，无须与游戏有关的具体信息，就学会了《乓》（Pong）、《太空侵略者》（Space Invaders）等多种不同游戏的玩法。

不管 DeepMind 背后站着谁，是 Elon Mask 还是 Google，DeepMind 目前都算是 AI 界的先锋之一，但在它之前呢？整个深度学习，或者说神经网络复苏的推动者有哪些呢？

第二个片段

2015 年 5 月 28 日，LeCun、Bengio 和 Hinton 三位人工智能大佬在 *Nature* 上刊发了一篇综述，题为 Deep Learning。

在计算机科学领域，很少有论文能在 *Nature* 和 *Science* 上发表，而这次 Nature 不但发了论文还专门为 Deep Learning 开了一个专栏，以综述形式一次性发表了多

篇高质量论文，涉及概率机器学习（恕我直译）、强化学习、机器人等多个方面。我觉得这已经可以标志深度学习带来了广泛的影响，且真正被学术界所接受。

具体学术成就我们先不提，先来看看这三位神经网络界的三巨头，“三巨头”的意思是，如果图灵奖颁给 Deep Learning 的改进和推广，而且可以给三个人的话（图灵奖一般给理论发明者），那么其中有一年就应该有这三位获奖。这三个人的关系很有意思，若论资排辈，自然是 Hinton→LeCun→Bengio。LeCun 是 Hinton 的博士后，当年 Michael Jordan 想去 Hinton 门下读博士后都被婉拒，而 Bengio 又是 Jordan 的博士后。这三人虽然有些师爷→师父→弟子的辈分，但彼此倒是心心相印。因为总是看到他们一起谋划一些大事。这些大事有意无意地推动了神经网络（深度学习）的崛起，他们三个人的坚持，正是神经网络今天又被推到风口浪尖的原因。

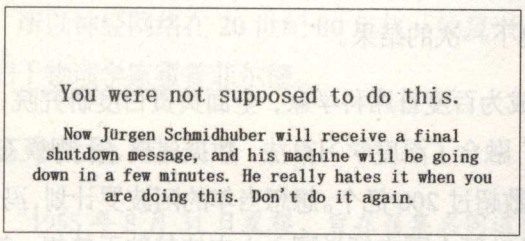
Hinton 的中文译名杰弗里·辛顿（Geoffrey Hinton）是反向传播算法和对比散度算法的发明人之一，也是深度学习的积极推动者，2006 年他的一篇文章开辟了这个新领域。经过他改进的算法能够对七层或更多层的深度神经网络进行训练，这让计算机可以渐进地进行学习。随着层次的增加，学习的精确性得到提升，同时该技术还极大地推动了非监督学习的发展，让机器具备“自学”的能力。辛顿于 1970 年在英国剑桥大学获得文学学士学位，主修实验心理学。此后于 1978 年在爱丁堡大学获得哲学博士学位，主修人工智能。此后他曾在萨塞克斯大学、加州大学圣迭戈分校、剑桥大学、卡内基梅隆大学和伦敦大学学院工作。他是盖茨比计算神经科学中心的创始人。DeepMind 创始人就曾在这里学习工作过。

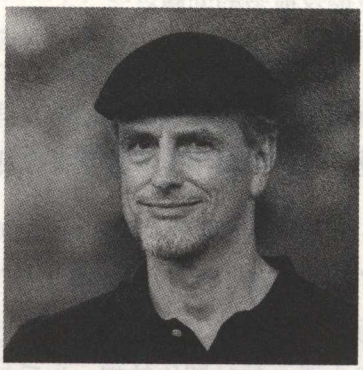
勒丘恩（LeCun）是法国人，已加盟 Facebook 的 AI 实验室，他最主要的贡献便是卷积神经网络（CNN）的改进再应用，虽说 LeCun 不是 CNN 的发明人，但他是第一个把 BP 算法用在 CNN 上，并且完善 CNN 使得它可以在一些真正的应用上工作的人，比如手写体，他也是自 1998 年之后近 20 年 CNN 的第一推动者。

左起依次为 LeCun、Hinton、Bengio 和吴恩达

除了这三位之外，不得不提两个人的名字。

一个是德国人 Jürgen Schmidhuber，他是 LSTM 的发明人，梯度消失的贡献人，也是递归结构的推动者。目前在瑞士人工智能实验室（IDSIA）担任研发主任。Schmidhuber 本人由于地处欧洲，和北美学术圈交流较少，另外本人性格比较内向，所以跟另外三位交集少，喜欢自己闷头搞研究。虽然名气没有这么大，但贡献不小。另外此人虽然内向，却不乏幽默，在他的个人主页 http://people.idsia.ch/~juergen/最下面有个红色按钮，这个红色按钮的上面有行字：不要按下红色按钮！当你忍不住按下后就会出现如下提示。

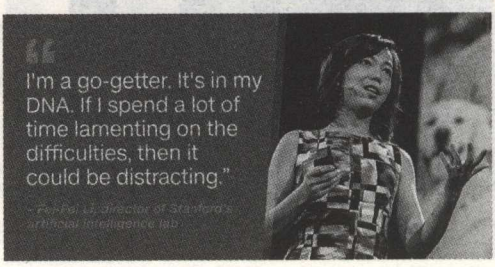




Jürgen Schmidhuber 教授

笔者所从事的生成模型里，基本上从 LSTM 开始文本生成模型的理论基础才稍显完善。

另一位是李飞飞，一位非常有“个性”的女性研究者，李飞飞有个特别的贡献就是收集了 ImageNet 库，这个库是用作视觉文本转换用途的。



李飞飞教授

第三个片段

2012 年 6 月，吴恩达和谷歌科学家合作，用 1.6 万台计算机搭建并模拟了一个人脑神经网络，向这个网络展示了 1000 万段从 Youtube 上随机选取的视频。结果这个系统在没有外界干涉的条件下，自己认识到“猫”是一种怎样的动物，并成功找到了猫的照片，识别率为 81.7%。“识别猫”成为深度学习领域的经典案例，奠定了吴恩达在人工智能领域的权威地位。

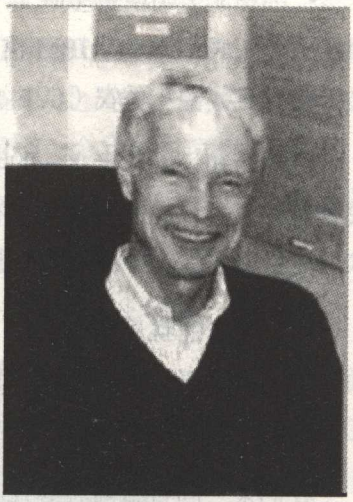
现在的人工智能可以做什么？对哪个行业影响最大？这是今年百度世界大会上被提及最多的问题。

吴恩达的答案是，“假如有一件事是一个正常人可以在 1 秒以下做到的，我们就可以使用人工智能自动做。假如你可以拿到一个具体重复发生的事情的海量数据，你就可以用这些数据来预测下一次的结果。”

吴恩达于 2014 年加入百度，成为百度首席科学家，全面负责百度研究院，参与“百度大脑”计划。“百度大脑”融合了深度学习算法、数据建模、大规模 GPU 并行化平台等技术，模拟神经元参数超过 200 亿个。想想当年的阿波罗计划，冯·布劳恩为了少年时期的梦想，先后鼓动两个国家倾举国之力支持其航天梦想，耗时十余年在美国打下了硅谷的基础，间接成就了今天的世界。正是因为不断有人追寻自身的梦想，并为之倾注毕生精力，人类才不断超越自己，不是吗？

第四个片段

1982 年，那时在加州理工担任生物物理教授的霍普菲尔德（J.Hopfield）提出了一种新颖的人工神经网络模型——霍普菲尔德神经网络。这个网络引入了能量函数的概念，使神经网络的平衡稳定状态有了明确的判断方法，是一个非线性动力学系统。霍普菲尔德神经网络是一种递归（复发型）神经网络，是一种结合存储系统和二元系统的神经网络。



霍普菲尔德

霍普菲尔德神经网络分成以下两个部分。

（1）离散的 Hopfield 网络用于联想记忆。

（2）连续的 Hopfield 网络用于求解最优化问题。

霍普菲尔德神经网络不仅对人工神经网络的信息存储和提取功能进行了非线性数学概括，提出了动力方程和学习方程，还对网络算法提供了重要公式和参数，使人工神经网络的构造和学习有了理论指导。1984 年，霍普菲尔德用模拟集成电路实现了自己提出的模型。使得大量学者又被激发起研究神经网络的热情，积极投身于这一领域。

所以神经网络在 20 世纪 80 年代开始复兴，除了时运之外，起码有一半功劳应归于物理学家霍普菲尔德。

第五个片段

“1955 年 8 月 31 日发起，旨在召集志同道合的人共同讨论“人工智能”（此定义正是在那时提出的）。会议持续了一个月，基本上以大范围的集思广益为主。”

——引自维基百科

- 约翰·麦卡锡（J. McCarthy）：当时是达特矛斯学院数学助理教授
- 马文·明斯基（M. L. Minsky）：哈佛大学数学与神经学初级研究员
- N. Rochester：IBM 信息研究经理
- 克劳德·香农（C. E. Shannon）：贝尔电话实验室数学家

可想而知，发起者之一的明斯基在当时的学术界只是一个助理教授。除了香农稍有名气外，其他人都还没有崭露头角，包括明斯基。所以这在当时无论是学术界还是普罗大众中都没有引起相关的重视。

第二年，也就是 1956 年，达特茅斯夏季会议中人工智能的概念被确定下来。而今年就是此次会议的 60 周年，普遍意义上讲，1956 年的会议被大家认为是人工智能的起点，所以 2016 年也是人工智能的 60 周年。

我们来一同看看当时的几个发起人。提到人工智能就不能避开明斯基，1951 年，马文·明斯基设计并构建了第一部能自我学习的人工神经网络机器，名为 SNARC。SNARC 是世界上第一个神经网络模拟器，是人工智能最早的尝试之一。1952 年，他发明会自行关闭电源的无用机器（Useless Machine）。

1955 年，他与麦卡锡、香农等人一起发起并组织了成为人工智能起点的具有历史意义的会议“达特茅斯会议”。1958 年，明斯基从哈佛转至麻省理工学院，同时麦卡锡也由达特茅斯来到 MIT 与他会合，他们在这里共同创建了世界上第一个人工智能实验室（MIT 计算机科学与人工智能实验室的前身）。

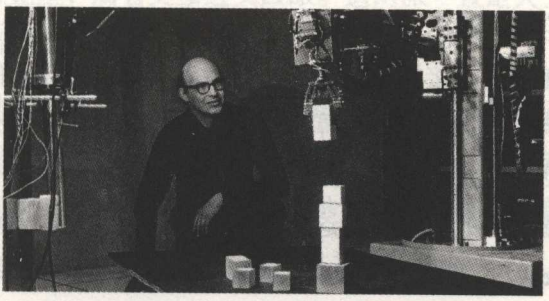
当然，明斯基是人工智能的发起人之一，也在神经网络方面做出了很大的贡献，他在 1954 年获得博士学位时，博士论文就是有关神经网络和脑神经的问题。但这并不代表他是人工神经网络的一贯追随者。

明斯基有个中学同学叫罗森布拉特，他在 1957 年提出了一个感知机的模型，这实际上就是一个单层的神经网络。明斯基在一次会议上和罗森布拉特大吵，他认为神经网络不能解决人工智能的问题。

接着，明斯基为了证明一个收敛性和泛化性，利用数学来研究神经网络，所以和派珀特合作了出了一本书，就是那本影响巨大、争论无比的书《感知机》

（Perceptrons）。这本书发表以后起到了相反的作用，神经网络走入了低谷，全世界都把原来的神经网络项目取消了，说白了就是一听你是研究神经网络的，对不起，没有经费，而今天的投资界一听到你是搞人工智能或深度学习的，那获得投资的机会大大增加，不知道明斯基、罗森布拉特的在天之灵会不会哭笑不得。为什么？因为马文·明斯基在他这本书里指出感知机这样一个异或的功能实现不了，而异或是非常基础的功能，如果这个问题都解决不了，那神经网络的计算能力实在有限。

这时候，历史静悄悄地等待着神经网络学科十多年后的大逆袭。



马文·明斯基

第六个片段

1959 年，两位美国工程师维德罗（B.Widrow）和霍夫（M.Hoff）提出了自适应线性元件（Adaptive linear element，简称 Adaline）。它是感知机的变化形式，也是机器学习的鼻祖模型之一。它与感知机的主要不同之处在于，Adaline 神经元有一个线性激活函数，允许输出是任意值，而不仅仅只是像感知机中那样只能取 0 或 1。自适应线性元件的主要用途是线性逼近一个函数式而进行模式联想。因为同感知机的原理相似，罗森布拉特享受盛誉时，维德罗也沾了光。后期他主要从事集成电路方面的工作，没有继续在人工智能领域工作。

感知机的暂时失败导致神经网络研究走入低谷，明斯基在《感知机》一书再版时，删除了原版中对罗森布拉特的个人攻击的句子，并手写了“纪念罗森布拉特”（In memory of Frank Rosenblatt）。美国电气电子工程师协会（IEEE）于 2004

年特别设立了罗森布拉特奖，以奖励在神经网络领域的杰出研究。世事都是轮回的，我们的老祖宗早就教导，得意时不要张扬，失落时不必萎顿。

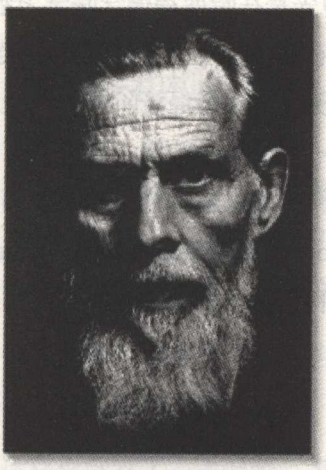
何为智能，我们看那星星之火

拥有传奇一生的阿兰·图灵。如果他还活着，今年应该已是 106 岁了。1949 年，图灵成为曼彻斯特大学计算机实验室的副主任，负责最早的真正的计算机——曼彻斯特一号的软件工作。他发表了一篇题为《计算机器和智能》（Computing Machinery and Intelligence）的论文，在这篇论文里，图灵第一次提出“机器会思考吗？”（Can Machines Think?），提出了“机器思维”的想法。他逐条反驳了机器不能思维的论调，做出了肯定的回答。同时，他尝试定出一个决定机器是否有感觉的标准：在彼此不接触的情况下，对话者通过一种特殊的方式（如具有电报通信功能的打字机等）与一台计算机交流，也就是进行一系列的问答。如果在一段时间内，对话者无法根据这些问答判断对方是人还是计算机，那么就可以认为这台计算机具备人类的智能，即这台计算机是能思维的。这就是著名的“图灵测试”（Turing test）。图灵预言 2000 年就可以出现骗过 30%裁判的机器人。也正是这篇文章，为图灵赢得了一顶桂冠——“人工智能之父”。可以说图灵确实牛，笔者从心底里佩服他。在 1947 年图灵就已经阐述了如何对机器学习的结果进行检查的方法，而且这一方法是很有远见和可操作性的！

图灵

自图灵提出“机器与智能”后，主流观点分成了两派，一派认为实现人工智能必须用逻辑和符号系统，这一派是自顶向下的。什么意思呢？我们在大学都学习过统计学，这个逻辑和符号就有点狭义统计学的意思，可以说是预设和反馈。还有一派认为通过仿造大脑可以达到人工智能，这一派是自底向上的，他们认定如果能造一台机器，模拟大脑中的神经网络，这台机器就有智能。这两派的最大区别是，前一派想搞清楚如何思考，将思考的过程演化成逻辑，姑且叫他们“思路模拟”派，也叫控制派；后一派可以称为仿生派，想从模拟生物学大脑入手，让大脑自由形成意识。估计他们的思想来源于中国古代的原始思维，套用两个哲学词，前者属于理论型，后者更接近实践派。这两派一直是人工智能领域里的两个阶级，在两条路线中斗争，这斗争有时还“你死我活”。

西方科学界是很高产的，环境好，能源源不断地产出各种新思想和解决方案。实际上，在仿生派与图灵提出“图灵测试”之前，1943 年，传奇人物麦卡洛可（McCulloch）和皮茨（Pitts）就发表了模拟神经网络的原创文章。下面简单聊聊传奇人物为什么传奇。



麦卡洛可

皮茨

沃尔特·皮茨（Walter Pitts）出生在底特律的一个困难家庭。皮茨从小就喜欢数学和哲学，初中时就读过罗素的书——《数学原理》。有兴趣的读者可以看

看罗素的《数学原理》，反正我是看不懂，皮茨同学当时才初中！皮茨读完这本书后估计是仰慕罗素，还给罗素写信。罗素爱才，邀请他到英国当他的研究生。但皮茨当时只有 12 岁，家里非常穷困，英国留学自然是不可能了。皮茨 15 岁时，他爸爸强行要他退学打工赚钱，皮茨一怒就离家出走了。

后来，皮茨听说罗素要到芝加哥大学访问，就只身来到芝加哥，还真见到了罗素，罗素把他推荐给当时也在芝加哥任教的卡尔纳普。卡尔纳普想看看这孩子到底有多聪明，就把自己写的《语言的逻辑句法》（感兴趣的读者可以自己去搜）送给皮茨，没过一个月，皮茨就看完了，把写满笔记的原书还给卡尔纳普。卡尔纳普惊为天人，于是给皮茨在芝加哥大学安排了一份打扫卫生的工作。可能有读者会问，为啥都惊为天人了，只给打扫卫生的活？我想老卡主要是两层用意，第一，扫马路至少可以避免流浪街头，有份固定工作，解决吃饭问题，皮茨是穷孩子啊！第二层用意估计是要再观察下皮茨同学，毕竟他不是科班出身。皮茨后来结识了也在芝加哥的麦卡洛可。

沃伦·麦卡洛克（Warren McCulloch）出生于美国东岸的一个优越家庭，他的家人全都是律师、医生、神学家或者工程师。他先是在哈弗福德学院学习数学，又去耶鲁进修哲学和心理学。1923 年，麦卡洛克在哥伦比亚大学学习实验美学，还取得了神经生理学的医学学位。麦卡洛克和皮茨来自社会经济阶层的两级，但他们一起创造了第一个关于精神的机械论理论——神经科学的第一种计算方法，现代计算机逻辑设计及人工智能的支柱。他们合作的成果就是神经网络的第一篇文章：《神经活动中思想内在性的逻辑演算》（A Logical Calculus of Ideas Immanent in Nervous Activity），发表在《数学生物物理期刊》上。后来这篇文章成了控制论的思想源泉之一。

1 神经网络是个什么东西

时间进入 2014 年，我们在各种媒体上看到与大数据和人工智能相关的新闻，其中炒得最热的是深度学习，深度学习是什么？神经网络跟深度学习是什么关系？我们带着这样的疑问来看如下事例。

1.1 买橙子和机器学习

有一天，你到水果店去买橙子，当然要挑选最甜、最熟的。你是根据橙子的重量来付钱的，而不是根据橙子的甜度或者成熟度，虽然水果店有时候会把好的橙子挑出一堆单独涨价，但是这里没这么做。

你妈妈曾经告诉你，橙子要挑深橙色的，颜色有点发红的，这样的橙子最甜，不要挑那些浅黄色甚至发青的，那些还没熟透。

这样你就有了一点经验，虽然这点经验是别人直接教给你的，但你也可以在挑选的时候用上：深橙色的橙子熟透了，甜。你在水果店，挑了些深橙色的橙子，称重付钱回家了。这事就这么完了？别急，还有下文。

小事儿也没那么简单。

你回到家，高高兴兴吃橙子，但是你发现并不是每个橙子都那么甜，有一些还是不够甜。看来妈妈的经验还是不足，只通过颜色判断橙子甜不甜，不是很靠谱。

你开始回忆自己吃过的橙子。到底什么样子的橙子最甜？好像是个头大而且是深橙色的比较甜，那些个头小的深橙色的橙子，大概有一半是不甜的。（比如，买了 100 个深橙色的橙子，有 50 个大的，都是甜的；另外 50 个小的，其中有 25 个是不甜的。）

你总算总结出了一条经验规则：大的深橙色的橙子是最甜的，哈哈。你又高高兴兴地去买橙子。可你熟悉的那种橙子卖完了，现在卖的是另一个品种，产自不同的地方，你之前总结的经验可能行不通了。你不知道之前的经验能不能迁移过去（迁移学习），于是你重新尝试，把各种橙子买回家尝，几次之后你发现这个品种中小的、浅黄色的橙子是最甜的！

过了几天，表妹来你家玩，她想吃橙子，于是你们一起去买。她不在乎橙子甜不甜，只要多汁就好。唉，从前总结的经验又不管用了。你只能开始新的一轮实验，目标就是多汁的橙子（优化目标变了）。你又总结出，捏起来越有弹性的橙子汁越多。

后来，你出国读书，国外的橙子跟你家乡卖的又不一样了，这里果蒂小、皮细腻的最好吃。毕业后，你结婚了，老婆不喜欢吃橙子，喜欢吃香蕉。生活是两个人的事情，从现在开始，你不光要买橙子，还要买香蕉！你积累的挑选橙子的经验规则都行不通了。你只能从头开始新的实验尝试，虽然这个过程很枯燥，但是你去做了，因为你爱她。

1.1.1 规则列表

你想把如何挑选橙子、香蕉等这些水果的方法和经验用程序实现，这样用计算机，甚至用你的手机摄像头扫一下，就能自动挑选出很多好吃的水果。因为你

积累了一些规则，可以这么实现：

```
if (color is deep orange  and size is big  and sold by favorite vendor):
orange is sweet.
if (flexible): orange is juicy.
etc.
```

这些规则越来越多的话，特征之间的组合就越来越麻烦和复杂，管理和使用都很麻烦。写程序实现的时候，谁会笨到写这么多 If...Then，这个时候就涉及我们的下一个问题。

1.1.2 机器学习

机器学习算法是普通算法的进化，更加聪明和自动。现在，我们分析如何把选橙子的问题定义成标准的机器学习问题。

随机选择一个市场上的橙子，作为我们要研究的目标（Training Data）。你可以用一个表格描述橙子的属性和类型的关系，每一行可以放一个橙子的数据，包括橙子的各种物理属性（Feature）：颜色、大小、形状、产地等，还有品尝时橙子的属性（Output Variables）：甜度、成熟度、多汁度等。现在这就是一个多分类问题，或者是回归问题，自动从数据中学习出特征与橙子类型的各种关系等。

如果用决策树算法，那么这个模型的样子就是你的规则库。当然，你也可以使用其他模型，比如线性模型，这样就是特征的线性组合了。

下次你去买水果，采集了一个橙子的各个指标特征，扔进你的模型，模型就会告诉你这个橙子的各种属性。

甚至你选择橙子的模型稍微变化下就可以选择香蕉了，这就叫迁移学习。

甚至你的模型会随着新的样本、新橙子的种类，变得越来越好，越来越全面，增量学习。

……

这就是机器学习，大家有点感觉了吗？

1.2 怎么定义神经网络

神经网络，是机器学习的一个分支，学名应该叫人工神经网络，与之相对应的是生物神经网络（Biological Neural Networks，BNN），我们将模拟生物神经网络的数学模型统称为人工神经网络模型，简称人工神经网络或者神经网络。

生物神经网络一般指生物的大脑神经元、细胞、触点等组成的网络，用于产生生物的意识，帮助生物进行思考和行动。其中重要组成部分是神经元，什么叫神经元呢？神经元怎么组成神经网络？单个神经元能产生什么价值？

这是我们要探讨的第一个问题，但在探讨之前，我们要看看人类大脑的有关神经是如何处理物体和认知物体的。

1.3 先来看看大脑如何学习

我们在学习、思考和识别事物时，基本原理比较简单：就是一个线路连接问题。大脑能做的每件事，能记住的每件事，都是很多细胞连接起来以后发挥的功能。人在刚出生时，脑内神经细胞之间的连接相对较少——只有哭泣、惊讶、呼吸、吸吮、吞咽等活动，以及完成仅有的无条件反射功能所需要的神经连接。孩子在第一次站起来直立行走，第一次说出一个清晰的音节，第一次对一个短语有所回应的时候，都完成了一次大脑的洗礼。对于整个人生来说这些第一次都很有意义，而对于大脑来说也是至关重要的一种“进化”，为了完成这些“超级任务”，大脑在无比努力地工作，脑内的神经元形成了无数条新的连接。

这些连接是怎样形成的呢？答案非常简单：是在与环境交互作用的过程中，在个体大脑内部形成的。在这个脑内“重新连线”的过程中，大脑通过感觉器官接收信息，在脑内进行信息加工，然后输出信息，产生反应，大脑通过一遍又一

遍地重复这个过程，逐渐积累，形成个体当前所具备的能力和创造性。

大脑的学习过程包括以下三个基本步骤或系统。

（1）信息输入。

（2）模式加工。

（3）动作输出。

1.3.1 信息输入

按照从开始思考、学习，到控制自己行动的顺序，大脑首先要从体内或体外接收信息。这类信息是通过身体的信息输入系统传入的。这个信息输入系统由我们从小就知道的五种基本感觉——视、听、嗅、触、味觉组成，还包括其他几种你可能不太熟悉的日常基本功能，如前庭系统，是通过复杂的内耳结构控制身体的姿势；本体感觉系统，是通过关节和肌肉内微小的神经感受器，告诉你肌肉和四肢的位置，以及运动情况；内感受系统，可以将机体内部器官的活动情况反馈给你。

每种信息输入系统在个体的学习和发育中都具有非常重要的作用。实际上，人的思考、推理、记忆和理解能力都始于、并且最开始就完全依赖于这些信息输入系统。

1.3.2 模式加工

信息输入系统的功能是基础，对于个体的学习必不可少。但是，只有这一个系统是不够的，我们还需要对外界输入的信息进一步加工，形成准确的经验。

信息通过输入系统——也就是我们的各种感觉系统进入大脑内部不同的处理中心。随后，这些处理中心对进入的信息进行分析和模式识别。当处理中心觉察出某种模式时，将它编译后存储到脑内相应的记忆区。比如第一次看到猴子，大脑视觉皮层至少有 30 个不同的区域会参与到这个过程中，每一块区域负责处理图

像的一方面，比如皮毛、尾巴、面部特征和动作等，这些综合起来形成了猴子的一个完整形象，储存到大脑的记忆区中。

以后，每当我们接触到这个模式，比如每次看到猴子的时候这个神经回路都会被加强，每个处理区块之间的连接会变得更稳定、更高效。对它的记忆将变得越来越牢固，回忆起来也越来越容易。

一旦我们的大脑学到了某种特定的信息模式，无论什么时候我们再遇到它，哪怕它有一定的变化，以其他不完全相同的形式出现，我们都应该能够再识别出来。每次接收新的信息时，我们就会重复该模式加工过程。这样一次又一次之后，我们的大脑里就会储存越来越多的信息模式。比如，多次见到猴子后，只要看到猴子的一部分，就能够准确地判断出：这是一只猴子。

当我们思考时，随时都可以从记忆库中回忆出这些信息模式。大脑思考得越多，就会发现已经储存的模式中关联越多，发现的关联越多，我们就开始理解信息的分类规则，以及各种细微的差别。如此继续下去，各种信息模式积累增多，就形成了我们对于这个世界的清晰、准确、有着各种关联的模型。

1.3.3 动作输出

当然，学习不仅与观察和理解有关，也与行动有关。但是，在你能够以创造性的、成功的方式行动之前，你必须对自己在做什么有想法，那就需要一份详细而精确的计划。怎样制订这样的计划呢？要遵循一种模式。值得庆幸的是，我们有一个充满各种模式的完整的记忆系统可以帮助我们制订计划。

制订动作输出的计划，需要预测该怎样做才可能奏效，这就需要运用我们过去形成的、关于这个世界的模型。换句话说，记忆不仅针对过去，对过去、现在和未来都很重要。

我们举个例子说明这个过程在孩子的大脑里会有什么发生：知识的学习掌握，是某种程度上记忆构建的过程，比如现在小孩子一岁，你要教他学习数字了，首先你要让他记住阿拉伯数字 1 是怎么写的，你要让他看这个形状，然后小家伙就

记住了 1 这个图像，那 1 代表什么含义呢？要用另外的方法再刺激，比如说给他看一个苹果、一只小熊、一个人等，他就在大脑中建立了 1 这个形状与苹果、小熊、人的联系，我们称这个联系的建立是神经元轴突终端连接到其他神经元上了，这种连接是生物学上的链接，这在解剖学中可以观测到。这个 1 的形状和苹果、小熊、人的形状存储在不同的神经元中或海马体中。通过对视觉神经的刺激、视觉神经元和记忆神经元的联系，我们完成了这个孩子学阿拉伯数字 1 的训练。（注意：为了简化说明整体记忆流程，这个过程的描述不是很精确。）图 1-1 所示为大脑区块位置图。

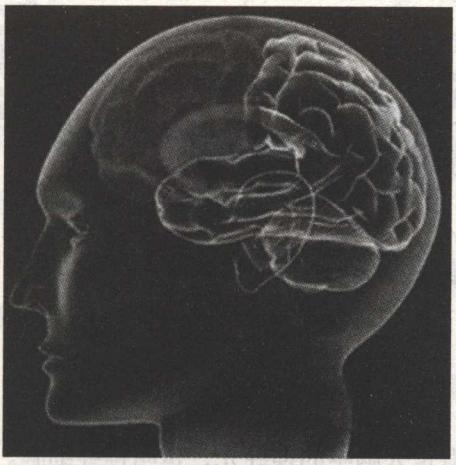


图 1-1 人的大脑

1.4 生物意义上的神经元

“神经元”（Neuron）事实上就是“神经细胞”（Nerve Cell）的别称。

1.4.1 神经元是如何工作的

图 1-2 所示为生物神经元的基本结构。如果把大脑比作一间工厂，这个工厂干的事情就是接受货物和送出货物，这个工厂处理接受到的货物，并决定送出去

的是什么货物。这些神经元就能对这些货物进行打包、拼装并储存起来；有时会将另外一些已经储存的货物打包或拆散送出。如何处理这些货物呢？我们可以想象这个工厂里有很多人员，有负责进货处理的，有负责出货处理的，决定什么货物送出、决定如何送出，这不是由其中一个人完成的。这跟公司的组织形式很像，工厂划分了很多部门，人们在各个部门中处理相应的货物。这个工厂非常特殊，每个在里面工作的人都可以互相组织形成一个新的部门，负责新的货物处理方式。

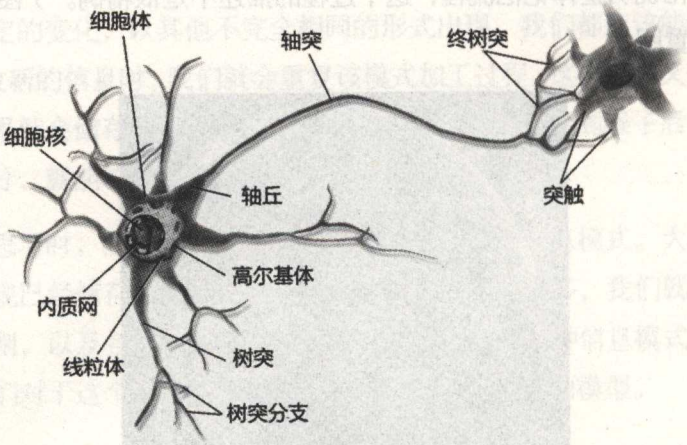


图 1-2　神经元的构成

工厂里的“人”就是大脑中的神经元，也叫神经细胞。神经元是神经系统结构和功能的基本单位。神经系统主要由神经元和神经胶质组成。神经元的形状和大小不一，但多数神经元具有一些共同结构。大致可以分为细胞体和突起两部分。胞体的中央有细胞核。细胞核是细胞的能量中心。通过化学反应，胞体为神经活动提供能量，并大量制造用于传递信息的化学物质。自胞体伸出两种突起：呈树枝状的被称为树突，它接收其他神经元的信息并传至胞体；那一根细长的突起称为轴突，它把冲动由胞体传至远处，传给另一个神经元的树突或肌肉与腺体。髓鞘由胶质细胞构成，包裹在轴突上，起着绝缘作用。一个神经元的轴突有许多分支末梢膨大，呈葡萄状，称为突触小体，它是传递信息给另一个神经元的发放端。

这里对神经元的结构描述可能有些枯燥，大家要注意几个关键词，因为我们

以后讲人工神经元时就知道：人工神经元是模拟了生物神经元的仿生工程。

细胞轴突是一条长长的纤维，它把细胞体的输出信息传导到其他神经元。树突用来接收其他神经元的输入信号。细胞体接收所有树突的输入，并通过细胞体内复杂的化学变化，确定细胞体是否需要对轴突产生输出。

神经元具有两个最主要的特性，即兴奋性和传导性。神经元的兴奋性具有一种很特殊的现象，当刺激强度未达到某一阈限值时（限值的概念为人工神经元模仿提供了理论依据，传输函数中大多数函数都是依据此原则来输出的），神经冲动不会发生，而当刺激强度达到该值时，神经冲动发生并能瞬时达到最大强度，此后刺激强度即使再继续加强或减弱，已诱发的冲动强度也不再发生变化。请大家深刻理解这个特性，兴奋性的原理解释了我们为什么需要传输函数，通过了解神经元的构造也可以推出传输函数的数学构成。

神经元的功能表现多种多样，归纳起来可分为三类。

（1）感觉神经元（传入神经元），其树突的末端分布于身体的外周部，接受来自体内外的刺激，将兴奋传至脊髓和脑。

（2）运动神经元（传出神经元），其轴突达于肌肉和腺体。运动神经元的兴奋可引起它们的活动。

（3）联络神经元（中间神经元），介于上述两种神经元之间，把它们联系起来或组成复杂的网络，起着神经元之间机能联系的作用，多存在于脑和脊髓里。

我们从幼儿认知世界来看神经元存储数据的方法。2 岁左右的幼儿对于火这个东西没有任何认知能力，有一天他偶然看到了蜡烛，然后让妈妈给他点着蜡烛，他很吃惊地看着蜡烛上的火光，既好奇又紧张（与蜡烛保持一定的距离，并紧紧地抓住妈妈的手）。妈妈鼓励他在注意安全的前提下靠近火光，感到热量。幼儿又问这是什么，回答这是“火”。（实际上，在人类的漫长遗传演化中，人从出生就有着敬畏火的记忆，这是遗传性在起作用，这里为了说明过程，不考虑遗传等因素。）

神经元在整个过程中是怎么配合并记录对“火”的认知的呢？首先，眼睛捕捉到相应的火的信号后会将信号传递给视神经（感觉神经元、传入神经元），视神经也是一种神经元，接受到的是外界刺激，这个刺激就是来自自然界的反射和散射的光线，由视神经传递给脑神经元（联络神经元），多次看到之后，其中一个神经元将火的形象存储下来。接下来，皮肤（感觉神经元）感觉到热，将热这个信号传递给脑神经，并且和刚才的视觉形象建立了联系；孩子如果离火太近感觉到很热时，就会将手缩回来（运动神经元）。

这种建立是双向的，例如，如果幼儿日后看到黄色发光物体，会认为它很热；如果闭着眼睛摸着很热的东西，那脑海中肯定反映出火的视觉形态。如果没有其他信息干预的话，甚至会产生一看到类似火的东西就把手缩回来的条件反射。

我举的这个例子不是很严谨，但足以让大家明白神经元的工作原理。接下来，我们看看神经元是如何组成神经网络的。

1.4.2 组成神经网络

神经元之间的连接不是固定不变的，在人的学习和成长过程中，一些新的连接会被逐渐建立起来，还有一些连接可能会消失。外界刺激就是神经网络的输入，在接收刺激后，刺激信号将传递到整个网络中，影响所有的神经元状态，神经元之间彼此连接并相互制约影响，不断调整彼此间的连接强度，直到达到稳定状态，并最终对刺激做出反应。神经元之间的关系变迁形成了生物体的学习过程。

生物神经网络是由很多神经元相互连接的，神经网络系统是一个极为庞大又错综复杂的系统。虽然每个神经元都十分简单，但如此大量的神经元之间非常复杂的连接却可以演化出丰富多彩的行为方式。同时，如此大量的神经元与外部感受器之间的多种多样的连接方式也蕴含了变化莫测的反应方式。总之，连接方式的多样化导致了行为方式的多样化。

脑神经系统的主要组成部分如图 1-3 所示。人脑具有阶层结构，其中最复杂的部分是处于大脑最外层的大脑皮层。在大脑皮层中密布着由大量神经元构成的

神经网络，这就使它具有高度的分析和综合能力。它是人脑思维活动的物质基础，是脑神经系统的核心部分。

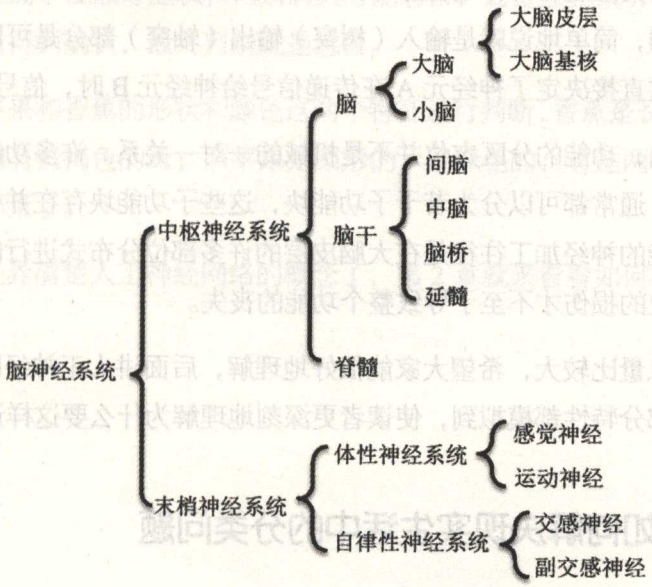


图 1-3 脑神经系统的主要组成部分

人们通过长期的研究，进一步探明了大脑皮层是由许多不同的功能区构成的。例如，有的区专门负责运动控制，有的区专门负责听觉，有的区专门负责视觉等。在每个功能区中，又包含许多负责某一个具体功能的神经元群。例如，在视觉神经区，存在着只对光线方向性产生反应的神经元。更进一步细分，某一层神经元仅对水平光线产生响应，而另一层神经元只对垂直光线产生反应（这个特性跟卷积神经网络有些关联度）。看到这里大家都觉得很熟悉，这不就是之前说的自组织（Self-Organization）工厂吗？这里需要特别指出的是，大脑皮层的这种区域性结构，虽然是由人的遗传特性所决定的，是先天性的，但各区域所具有的功能大部分是人在后天通过对环境的适应和学习而得来的。神经元的这种特性称为自组织特性。所谓自组织，就是每个神经元都自己决定和另外哪些神经元链接，甚至不链接，没有从一个“领导”的角色去安排大家的工作，而这种自己决定的特性

构成了神经元的自主学习(对应于神经网络模型的训练),不存在外部教师的示教。还应指出，神经元的这种自组织特性来自于神经网络结构的可塑性，即神经元之间相互连接的突触随着动作电位脉冲激励方式与强度的变化，其传递电位的作用可增加或减弱，简单地说就是输入（树突）输出（轴突）部分是可以由链接强度来表示的，这直接决定了神经元 A 在传递信号给神经元 B 时，信号是有强度的。

还应指出，功能的分区定位并不是机械的一对一关系。许多功能，特别是高级思维功能，通常都可以分为若干子功能块，这些子功能块存在并行关系。对于一个特定功能的神经加工往往是在大脑皮层的许多部位分布式进行的。正因为如此，某一部位的损伤才不至于导致整个功能的丧失。

本节信息量比较大，希望大家能很好地理解，后面讲人工神经网络时会把这节讲的绝大部分特性都模拟到，使读者更深刻地理解为什么要这样设计神经元。

1.5 大脑如何解决现实生活中的分类问题

人在学习知识，认识自然的过程中，有个基本的能力就是对已知事物进行学习，对已知或未知的事物进行分类，而对于人或物体的分类又是人类基于学习到的知识的一个逻辑推理过程。

让我们思考下面几个问题。

（1）人是如何进行垃圾邮件识别的？我们看邮件一眼，仅仅靠看标题就知道是垃圾邮件，这是如何做到的？

（2）医生是如何判断病人疾病的？

（3）我们怎么把香蕉和苹果分开？

大脑在做上面的推理时，当然涉及知识库的积累，如果有了这些知识，人在看垃圾邮件时，也许是瞥到了标题上出现的“财务”或“销售技能”，或者是看到了发送邮箱不符合正常人取名字的规律，就直接判别为垃圾邮件了。

而医生对于疾病的判断，这里不说判断，说归类更符合实际场景。医生对于疾病的诊断，除了望闻问切之外，主要是从基本的检查对疾病分类，比如先通过流鼻涕、喉咙痛、发热等症状，判别病人可能感冒，通过按压或听音排除其他炎症等，验血后得到确认，最终判别就是感冒。

我们对苹果和香蕉的形状和颜色这两个特征进行判断，香蕉是长且有小弯的，一般是黄色的有些褐色的斑；而苹果是圆形的、红绿色的，有这两种特征就很好区分香蕉和苹果了。

我们已经弄清楚人工神经网络的概念了，第 2 章就来看看如何构造一个基本的神经网络。

2 构造神经网络

本章我们来学习如何构造一个人工神经网络，当然这一切都要从基本单位人工神经元说起。

2.1 构造一个神经元

本节我们就来设计一个人工神经网络的基本单位——人工神经元。依照刚才学习到的单个神经元来创造一个简单的数理图形，能够反映现实神经元的基本模型，注意这里是基本模型，是从基本功能角度去模拟，而不是从单个蛋白细胞的角度去模拟。

前面我们说到生物神经元的时候，着重提了神经元的输入部分（树突）、处理部分（细胞体）和输出部分（轴突）。

我们先来看看信号的输入部分树突（如图 2-1 所示），这里我们用 p 代表树突，多个输入信号源，多个树突，分别是从 p_1 开始到 p_n；我们用 w 代表树突的强度权重。

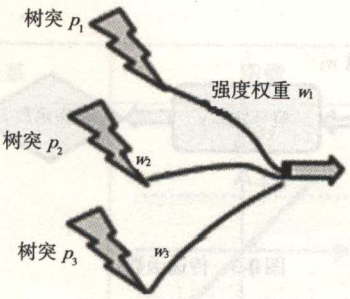


图 2-1 神经元的输入部分：树突

树突将收集到的刺激源往后传递，这里的刺激源可以是外界五感信号源，也可以将其他神经元细胞产生的结果(output)作为目前这个神经元的输入(input)，这里我们统一称作刺激源，如图 2-2 所示。

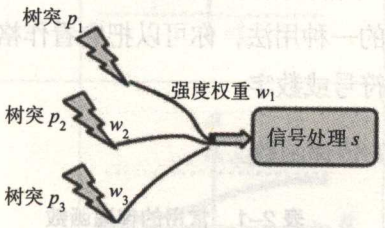


图 2-2 树突将信号传往信息处理机构（模拟生物层面上的细胞体）

在信号处理里面处理的是所有树突的信号源及相关强度的计算。

这种强度可以用以下简单公式来表示：

$$s = p_1w_1+p_2w_2+p_3w_3+\cdots\cdots+p_nw_n$$

从以上公式中我们可以看到，只是将输入的信号 p 乘以强度 w，然后依次累加得到 s。实际上，信号处理是归纳了所有神经元输入结果，并将其作为一个结果输出，这样就方便我们处理了。

以上就是构造人工神经元的基本过程，这也是根据生物神经元的原理做的算法模拟，即接受刺激信号并汇总输出。非常简单，对不对！仅仅这样，我们设计的这个神经元还无法干活，还要加点东西，如图 2-3 所示。

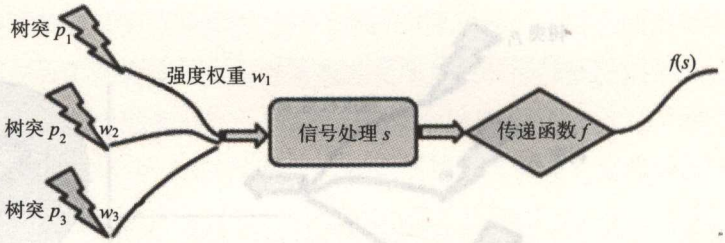


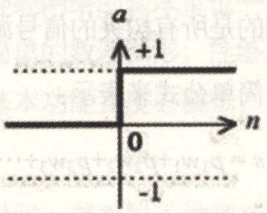
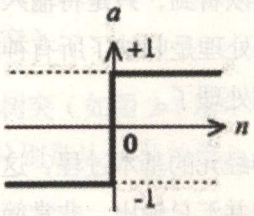
图 2-3　传递函数

我们要先理解什么叫传递函数?

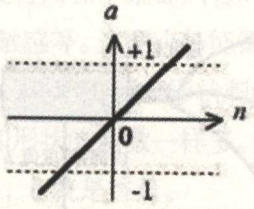
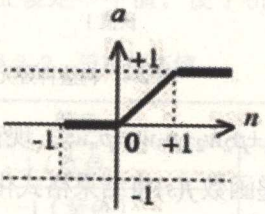
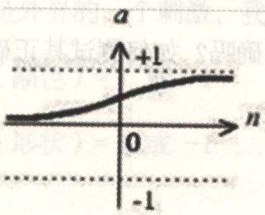
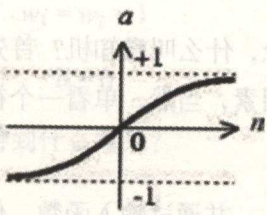
在信号处理这端，我们假设有一个输出结果 $s(n)$，这个结果有可能是 0 到 1 这个区间的一个数字，也可以是任意一个可能大于 0 或者小于 0 的整数，然而对于一个分类运算来说，大多数时候，我们需要输出一个 0 或 1，代表是或否，怎么处理呢？如果大于 0，我们就把它作为 1，如果小于 0，我们就把它输出为 0，这是最简单的传递函数的一种用法，你可以把它看作格式化输出结果，将结果变成我们可以使用的一种符号或数字。

表 2-1 所示为常用的传递函数。

表 2-1　常用的传递函数

函数名称	映射关系	图像	缩写	说明
阶梯函数	a=0，$n \leqslant 0$ a=1，n>0		Step	n 大于等于 0 时，输出 1，否则输出 0
符号函数	a=−1，n<0 a=1，$n \geqslant 0$		Sgn	n 大于等于 0 时，输出 1，否则输出 −1

续表

函数名称	映射关系	图像	缩写	说明
线性函数	$a=n$		Linear	n 本身就是神经元输出
饱和线性函数	$a=0$，$n<0$ $a=n$，$0\leqslant n\leqslant 1$ $a=1$，$n>1$		Ramp	n 小于 0 时，输出 0，n 在 0 到 1 区间时，输出 n，n 大于 1 时输出 1
对数 S 形函数	$a=1/(1+exp(-n))$		Sigmoid	有界函数，无论 n 如何，输出永远在(0,1)的开区间
双曲正切 S 形函数	$a=\dfrac{exp(n)-exp(-n)}{\exp(n)+exp(-n)}$		Tanh	有界函数，无论 n 如何，输出永远在(-1,1)的开区间

我们几乎完整地模拟了一个神经元的功能，为什么说几乎呢？

因为还有个非常容易漏掉的地方。

神经元细胞的处理我们用 s 模拟，我们只是简单地将信号做了一个加权处理，

而神经元本身的特性我们没有模拟出来，我们给 s 加个内置的处理输入源，并用 b 模拟这种内部的强度，如图 2-4 所示。

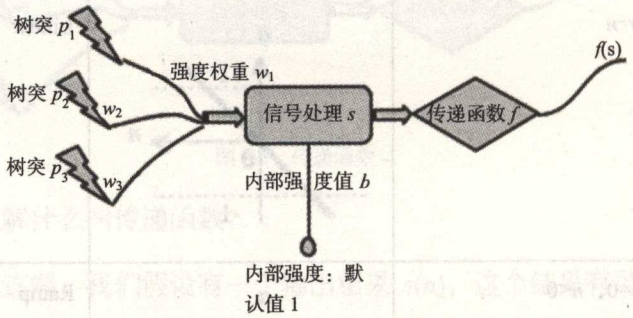


图 2-4　构造神经元

原来的公式是 $s=p_1w_1+p_2w_2+p_3w_3+p_nw_n$，现在的 s 函数是 $s=p_1w_1+p_2w_2+p_3w_3+p_nw_n+b\times 1$，然后将用传递函数 $f(s)$将结果格式化输出。

以上成功的模拟了生物神经元，很简单吧！

我们构造的神经元正确吗？如何测试其正确性呢？能用现实中的一个问题来检验吗？

2.2　感知机

我们先来看一个概念，什么叫感知机？首先，感知机是最简单的神经网络，它具备神经网络的必备因素，当然，单看一个神经元是无法理解“网络”的概念的。

网络接收若干个输入，并通过输入函数、传输函数给出一个网络的输出。这个网络已经可以解决苹果和香蕉的分类问题。本节将具体介绍其内部原理。

现在我们不叫它神经元了，我们叫它感知机，它是最简单的神经网络。

我们用感知机解决一个经典分类问题——如何分辨香蕉和苹果。

若你的年龄不低于 2 岁，那么我想你能分辨出香蕉和苹果，而计算机是如何做到的呢？

为了将问题简单化，我们假设香蕉和苹果都只有两个特征——颜色和形状，其他特征我们不考虑，比如气味、触感等。这两个特征都是基于视觉刺激得到的，用树突 p_1 代表输入颜色刺激状态，树突 p_2 代表形状刺激状态，权重 w_1 默认都设置为 1，即假设之前受到的颜色和形状的刺激一样多。再简化点，我们将内部强度 b 设为 0。我们定义 1 就是苹果，0 就是香蕉。

我们为苹果和香蕉的两个特征设定一个值，便于机器计算，如表 2-2 所示。

表 2-2　设定特征值

品种	颜色	形状
苹果	1（红色）	1（圆形）
香蕉	-1（黄色）	-1（弯形）

颜色和形状对神经元来说都是外界的一个刺激，我们说：

$$p_1（颜色）= 1 \text{ 或 } -1$$

$$p_2（形状）= 1 \text{ 或 } -1$$

预设：

$$w_1 = w_2 = 1$$

$$b = 0$$

代入之前 s 的那个公式看看得到什么结果？

对苹果的鉴别如下：

$$s = 1 \times 1 + 1 \times 1 + 0 = 2$$

对香蕉的鉴别如下：

$$s = -1 \times 1 + (-1) \times 1 + 0 = -2$$

我们通过输入苹果和香蕉的两类特征值得到不同的输出结果，也就是 s，苹果的输出结果是 2，香蕉的输出结果是-2。

几乎已经完成识别了。

我们定义的是 1 为苹果，0 为香蕉，如何将-2 或 2 变为 1 或 0 的分类表达呢？这就需要传递函数处理了，我们选用 *step* 函数作为数据格式处理函数，*step* 的输出就是 0 和 1，将数据代入，则 $step(2)=1$，$step(-2)=0$，和我们预期的一样。

我们利用简单感知机实现了一个识别苹果和香蕉的例子。

聪明的读者会发现，所有的结果都是基于设定好的相关参数 w_1、w_2 和 b 这些神经元关键权重参数。

我们为什么要取值为 1、1 和 0 呢？如果取其他值有什么问题吗？会对结果有什么影响吗？

2.3 感知机的学习

回答 2.2 节最后的问题：换一个值，结果可能完全不对，那我们是如何得到 1、1 和 0 的呢？这需要使用感知机的一套学习规则，保证我们随意取个权重参数也能使输出的值是正常的。

感知机的学习规则也是一种训练方法，目的是修改神经网络的权值和偏置。

$$w(new)=w(old)+ep$$

$$b(new)=b(old)+e$$

其中 e 表示误差，$e=t-a$，t 为期望输出，a 为实际输出。

下面我们用实例的方式推导感知机的学习规则。

例子，设 $w_1=1$，$w_2=-1$，$b=0$。

（1）苹果的形状和颜色均输入属性 1。得到：

$$\begin{aligned} s &= p_1w_1+p_2w_2+b \\ &= 1-1+0 \\ &= 0 \end{aligned}$$

$f=0$ （套用 *step* 函数）

（2）观察了结果，期望结果 1，实际得到了 0 这个结果，这里输出值错误了，我们利用感知机的学习规则计算误差。

$$\begin{aligned} e &= t-z \\ &=1-0 \\ &=1 \end{aligned}$$

我们得到了误差 $e=1$，再把值代入：

$$\begin{aligned} w_1new &= w_1old+ep \\ &=1+1\times 1 \\ &=2 \end{aligned}$$

$$\begin{aligned} w_2new &= w_2old+ep \\ &=-1+1\times 1 \\ &=0 \end{aligned}$$

$$\begin{aligned} bnew &= bold+e \\ &=0+1 \\ &=1 \end{aligned}$$

（3）使用新的权值带入感知机，重新计算苹果的属性输入。

$$\begin{aligned} net &= p_1w_1+p_2w_2+b \\ &=1\times 2+1\times 0+1 \\ &=3 \end{aligned}$$

$$f(step)=1$$

（4）纠正误差后，苹果判断正确。尝试判断香蕉。

$$net=p_1w_1+p_2w_2+b$$
$$=-1\times2-1\times0+1$$
$$=-1$$
$$f(step)=0$$

香蕉判断也正确，误差为 0，学习结束。

我们利用了感知机的（有监督）学习规则进行误差纠正，并把新的权值代入公式计算输出得到我们期望的值。

在有监督的学习规则中，我们能通过期望值不断修正权重，最终得到一个可用权重并用已经训练好的感知机去做一些事情。

2.4 用代码实现一个感知机

了解感知机的运作原理之后，如果你是程序员或者你习惯用代码去表示，那么现在就可以动手试试了！我们挑一个受众面比较广的语言作为基础语言——Java，构建一个感知机。

2.4.1 Neuroph：一个基于 Java 的神经网络框架

在整个第 2 章中，我们都将用 Neuroph 作为基础框架，实现本书提及的神经网络和相关案例。

下面，我们简单认识一下 Neuroph 神经网络的框架。Neuroph 是一个轻量级的开源 Java 神经网络框架，结构简单清晰，非常适合初学者入门，随着深入学习，可以慢慢了解 Neuroph 不只是初学者的入门工具，它可以简化神经网络的开发和实现，也能实现一些用在生产环境中的算法，图 2-5 所示为它的基本类库。

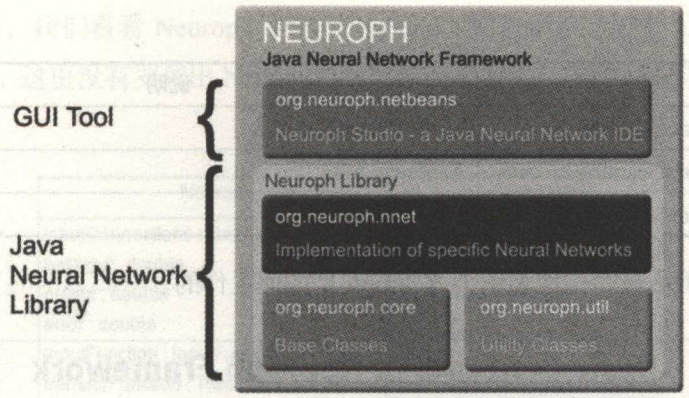


图 2-5 Neuroph 神经网络的框架

Neuroph 由两部分组成。一部分是由基于 Java 开发的 API 组成，你可以方便地利用这部分 API 创建神经网络。另一部分是图形工具，能直接通过简单的图形化工具构造一个神经网络，帮助我们构造多层神经网络时更加快捷方便。

Java API 部分分成三块，一块是 neuroph.core，这是 Neuroph 的核心库，如表 2-3 所示。

表 2-3 Neuroph 类库说明

类库	说明
org.neuroph.core	提供基础类库和基本的核心组件
org.neuroph.core.data	数据设置及相关计算
org.neuroph.core.events	神经网络的学习事件系统
org.neuroph.core.input	提供神经元输入功能
org.neuroph.core.learning	神经网络学习算法
org.neuroph.core.learning.error	为学习规则提供错误处理纠正
org.neuroph.util	提供了一组工具类：帮助构建神经网络、类型标识、向量分析等
org.neuroph.nnet	迅速构建神经网络
org.neuroph.nnet.comp	神经网络组件
org.neuroph.nnet.comp.layer	多种 layer 类型
org.neuroph.nnet.comp.neuron	多种特殊的 Neuron 类型

续表

类库	说明
org.neuroph.nnet.learning	基于神经网络的算法
org.neuroph.nnet.learning.kmeans	K-Means 算法
org.neuroph.nnet.learning.knn	KNN 算法

图 2-6 所示最简化地表示了 Neuroph 是如何工作的。

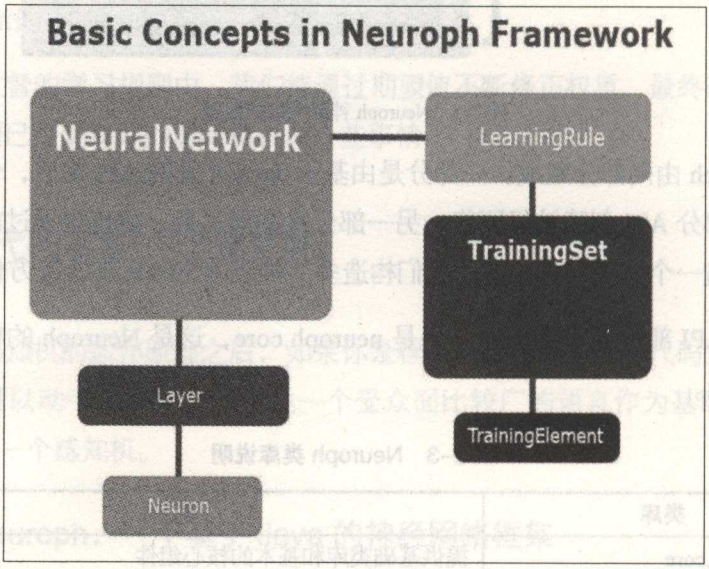


图 2-6 Neuroph 是如何工作的

（1）神经网络和学习规则对应，神经网络按照一定的学习规则训练相应的数据集。

（2）神经网络由基础的层（layer）组成，按照结构分为输入层、隐藏层和输出层，我们讲神经网络的时候会具体谈到各层的概念。

（3）神经网络的每层由最基础的神经元组成。

（4）训练规则包含一个训练集，允许有多个训练集，训练集由单个训练元素组成。

接下来，我们看看 Neuroph 中的类 Neuron，它表示单个神经元的构造。如图 2-7 所示，这里没有罗列出 Neuron 所有的属性和方法，只展示了与本节相关的部分。

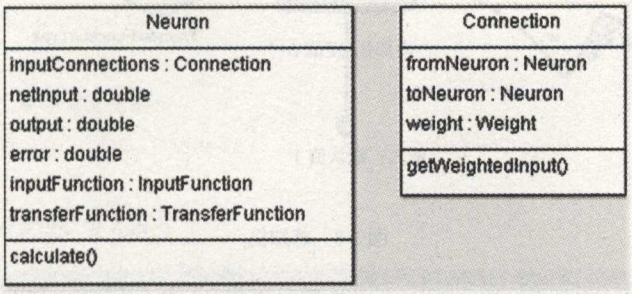


图 2-7 类 Neuron

其中 inputConnections 表示神经元的输入连接。比如，一个输入刺激到神经元，就构成一条输入。由于神经元是可以有多个输入的，所以神经元和输入连接是一对多的关系。netInput 表示净输入，输入函数的输出。output 表示神经元的输入。error 为神经元的误差。inputFunction 表示输入函数，通常选择加权求和。transferFunction 表示传输函数。

另一个重要的类为 Connection。它表示神经元的连接，可以是两个神经元之间的连接，也可以是输入信号与神经元之间的连接（实际上，输入信号可以看作一个输出永远等于输入的简单神经元），weight 表示这个连接的权重。

这个结构已经与之前所述的感知机原理很接近了，下面我们就来构造一个感知机。

2.4.2 代码实现感知机

回到本节的主题，我们用代码实现一个感知机，以便更加深入地了解感知机的构造。你也可以选择跳过所有与代码相关的章节，不影响阅读。

首先，我们先想想要构建一个什么样的感知机：我们将神经元输入链接个数、神经元输出链接个数和传输函数类型作为输入的参数，如图 2-8 所示。

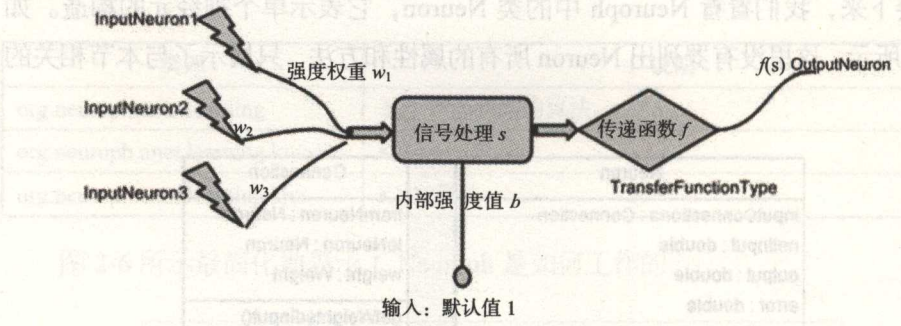


图 2-8　感知机

```
private void createNetwork(int inputNeuronsCount, int outputNeuronsCount,
TransferFunctionType transferFunctionType) {
  //设置神经网络类型，这里我们将类型设置为感知器
  this.setNetworkType(NeuralNetworkType.PERCEPTRON);

  //初始化神经元输入刺激设置
  NeuronProperties inputNeuronProperties = new NeuronProperties();
  inputNeuronProperties.setProperty("transferFunction",
TransferFunctionType.LINEAR);

  //创建输入刺激
  Layer inputLayer = LayerFactory.createLayer(inputNeuronsCount,
inputNeuronProperties);
  this.addLayer(inputLayer);

  NeuronProperties outputNeuronProperties = new NeuronProperties();
  outputNeuronProperties.setProperty("neuronType", ThresholdNeuron.
class);
  outputNeuronProperties.setProperty("thresh", new Double(Math.abs
(Math.random())));
  outputNeuronProperties.setProperty("transferFunction", transferFunctionType);
  //为 sigmoid 和 tanh 传输函数设置斜率属性
  outputNeuronProperties.setProperty("transferFunction.slope", new Double(1));
```

```
 //create一个神经元的输出
 Layer outputLayer = LayerFactory.createLayer(outputNeuronsCount,
outputNeuronProperties);
 this.addLayer(outputLayer);

 //在输入和输出层中建立全链接
 ConnectionFactory.fullConnect(inputLayer, outputLayer);

 //为神经网络设置默认输入输出
 NeuralNetworkFactory.setDefaultIO(this);
            this.setLearningRule(new BinaryDeltaRule());
}
```

至此，一个简单的感知机已经建立，调用这个感知机可以处理诸如水果分类之类的简单问题。我们将训练这个神经网络，让它具有记忆和解决简单问题的能力。

2.4.3 感知机学习一个简单逻辑运算

我们已经建立了一个简单的感知机，本节中我们将训练这个感知机，并让学习逻辑运算 AND。

首先，我们列出逻辑运算 AND 中的基本规则：

0 and 0=0

0 and 1=0

1 and 0=0

1 and 1=1

也就是说，在感知机接收 0、0 输入时，感知机应该响应 1；当接收 1、1 输入时，应该响应 1。当接收 0、1 或 1、0 时，应该输出 0。AND 主要用在条件判

断中，也就是当两个子条件都成立时，才往下进行，两个子条件就用 AND 连接。

我们先给出完整代码（对应的 neuroph 的版本号 2.7），再顺着代码的脉络看看感知机是如何学会 AND 运算的。

```
public static void main(String args[]) {

    //建立训练集，有两个输入一个输出
    DataSet trainingSet = new DataSet(2, 1);
    trainingSet.addRow(new DataSetRow(new double[]{0, 0}, new double[]{0}));
    trainingSet.addRow(new DataSetRow(new double[]{0, 1}, new double[]{0}));
    trainingSet.addRow(new DataSetRow(new double[]{1, 0}, new double[]{0}));
    trainingSet.addRow(new DataSetRow(new double[]{1, 1}, new double[]{1}));

    //建立一个感知机，定义输入刺激是 2 个，感知机输出是 1 个，这里我们调用 Neuroph 提供
的 Perceptron 类。
    NeuralNetwork myPerceptron = new Perceptron(2, 1);
    LearningRule lr =myPerceptron.getLearningRule();

    lr.addListener(this);
    //开始学习训练集
    myPerceptron.learn(trainingSet);
    //测试感知机是否正确输出，打印
    System.out.println("Testing trained perceptron");
    testNeuralNetwork(myPerceptron, trainingSet);
}
```

我们运行程序，结果如下：

```
[main] INFO org.neuroph.core.learning.LearningRule - Learning Started
1. iterate
2. iterate
3. iterate
4. iterate
5. iterate
```

```
[main] INFO org.neuroph.core.learning.LearningRule - Learning Stoped
5. iterate
Testing trained perceptron
Input: [0.0, 0.0] Output: [0.0]
Input: [0.0, 1.0] Output: [0.0]
Input: [1.0, 0.0] Output: [0.0]
Input: [1.0, 1.0] Output: [1.0]
```

结果输出表明，网络经过 5 次迭代后，误差为 0，初始的权值和偏置是随机数，接着网络根据输入的训练数据迭代学习，不断调整权值和偏置，并最终记忆 AND 逻辑运算。从测试数据中可以看到，4 个输出已经完全正确，该网络已经对 AND 逻辑操作有正确的响应。

同理，只需要改变训练数据，就可以让感知机记忆其他内容，比如 OR 逻辑运算。

0 or 0=0

0 or 1=1

1 or 0=1

1 or 1=1

我们依据 OR 逻辑运算的规则，更改上例的训练数据：

```
trainingSet.addRow(new DataSetRow(new double[]{0, 0}, new double[]{0}));
trainingSet.addRow(new DataSetRow(new double[]{0, 1}, new double[]{1}));
trainingSet.addRow(new DataSetRow(new double[]{1, 0}, new double[]{1}));
trainingSet.addRow(new DataSetRow(new double[]{1, 1}, new double[]{1}));
```

然后测试该网络，给出的输出可能为：

```
[main] INFO org.neuroph.core.learning.LearningRule - Learning Started
1. iterate
2. iterate
3. iterate
```

```
4. iterate
5. iterate
6. iterate
7. iterate
8. iterate
9. iterate
10. iterate
11. iterate
12. iterate
[main] INFO org.neuroph.core.learning.LearningRule - Learning Stoped
12. iterate
Testing trained perceptron
Input: [0.0, 0.0] Output: [0.0]
Input: [0.0, 1.0] Output: [1.0]
Input: [1.0, 0.0] Output: [1.0]
Input: [1.0, 1.0] Output: [1.0]
```

可以看到，经过 12 次迭代，网络已经正确记忆了 OR 逻辑运算。到这里，细心的读者可以发现，在学习 AND 逻辑运算的时候，网络进行了 5 次迭代，但在学习 OR 逻辑运算的时候迭代了 12 次。这种结果是完全随机的，实际上在网络建立的时候，初始权值是随机产生的（绝对值小于 1），因此迭代几次才能使网络给出正确输出是不确定的，但不论初始值如何，经过有限次迭代，网络一定能完全记住训练数据。

2.4.4 XOR 问题

大家还记得罗森布拉特吗？他是我们在第 0 章提到的那位神经网络中感知机奠基人之一，并且还是利用计算机模拟了感知机模型的天才，他是怎么被自己的中学同学明斯基找到感知机漏洞，郁闷至死的呢？对！就是因为 XOR 问题，我们来看看 XOR 能不能在我们的感知机模型中得到解决。

老规矩，先列出 XOR 的运算规则：

0 xor 0=0

0 xor 1=1

1 xor 0=1

1 xor 1=0

可以看到，当输入的两个值相等时，XOR 返回 0，否则得到 1。

依然使用给出的感知机学习 XOR 会得到什么结果呢？感知机还可以正常工作吗？

我们将训练数据替换成如下代码：

```
trainingSet.addRow(new DataSetRow(new double[]{0, 0}, new double[]{0}));
trainingSet.addRow(new DataSetRow(new double[]{0, 1}, new double[]{1}));
trainingSet.addRow(new DataSetRow(new double[]{1, 0}, new double[]{1}));
trainingSet.addRow(new DataSetRow(new double[]{1, 1}, new double[]{0}));
```

再来运行一下程序：

```
.....
573884. iterate
573885. iterate
573886. iterate
573887. iterate
573888. iterate
573889. iterate
573890. iterate
573891. iterate
573892. iterate
573893. iterate
573894. iterate
.....
```

我暂停了，已经计算了 60w 次仍然没有得到结果。

如果不手动终止程序，它将永远运行下去。读者如果执行此程序，一定也会得到一样的结果。这是什么原因呢？为什么感知机可以轻松地记忆 AND 和 OR 运算，但是无法学习 XOR 呢？

先不解答此问题，我们先来构造一个神经网络，大家且往下看。

2.5 构造一个神经网络

我们谈生物学上的神经网络时曾提到过，单个生物神经元有不同的作用，当这些不同种类的神经元依据某种结构联系起来时，就成为神经网络。人类靠这样的神经网络分析、判断、思考并指导肢体进行反应，其中联系或连接非常重要，我们模仿人类大脑，把人工神经网络看成是一种运算模型，由大量的节点（或称神经元）相互连接构成。每个节点代表一种特定的输出函数。每两个节点间的连接都代表一个通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出根据网络的连接方式，权重值和激励函数的不同而不同。

先看看单层神经网络，如图 2-9 所示。

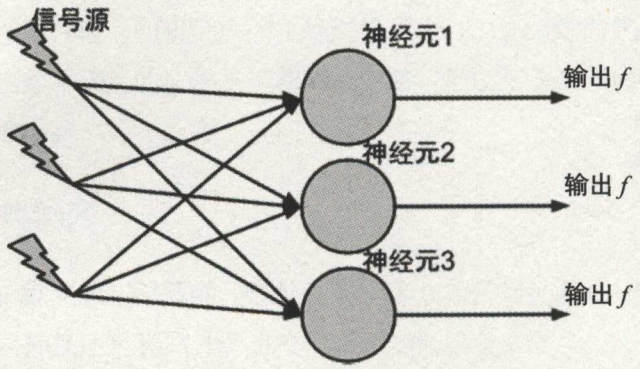


图 2-9　单层神经网络

在实际运用中，我们很少用单层神经网络直接解决问题，但它是学习多层神经元网络的基础。

2.5.1 线性不可分

我们已经学习了单个神经元、感知机和单层神经网络（又被称为单层前馈型神经网络）的知识。

我们已经可以让模型正确地学习AND逻辑运算和OR逻辑运算，如果你愿意，你甚至可以将之前说的香蕉和苹果分类，再回头看看表 2-1，你会发现这就是个逻辑运算。

将香蕉的-1 更换成 0，得到表 2-4。

表 2-4 逻辑运算

品种	颜色	形状
苹果	1（红色）	1（圆形）
香蕉	0（黄色）	0（弯形）

利用 AND 运算规则：

0 and 0=0 香蕉

1 and 1=1 苹果

等等，如果输入是以下组合怎么办？

0 and 1=0 香蕉？

1 and 0=0 香蕉？

这个计算式表示了无论｛颜色红色、形状弯形｝或｛形状圆形、颜色黄色｝都输出香蕉了！这明显和我们的期望不符。

怎么办，这就要先引入一个概念：线性不可分。

还是以上面的苹果和香蕉为例，如图 2-10 所示。

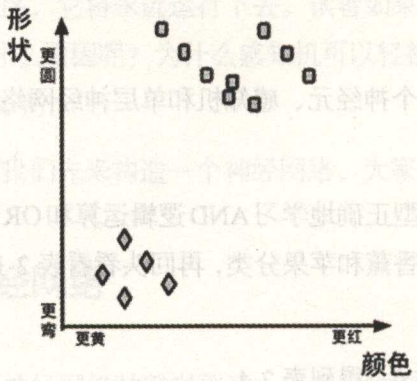


图 2-10　线性可分展示

我设计了一个能直观显示的图，大家一眼就可以看出方形的接近苹果的特征，菱形的接近香蕉的特征，这两种水果的特征如此鲜明，我们可以用一条直线分割这两类水果，如图 2-11 所示。

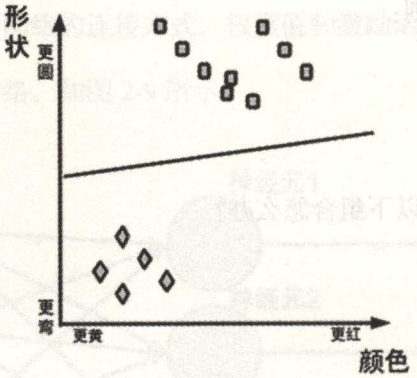


图 2-11　线性可分展示

这条直线变成了分割苹果和香蕉的最优直线，先不去探讨如何最优化得到这条直线。我们能够用直线清晰地分类这两类水果，这就是线性可分。

我们知道现实中很多东西都是线性不可分的，比如你去分类两类以上的水果，大多数情况下，我们不能用一条直线把多种水果分开，如图 2-12 所示。

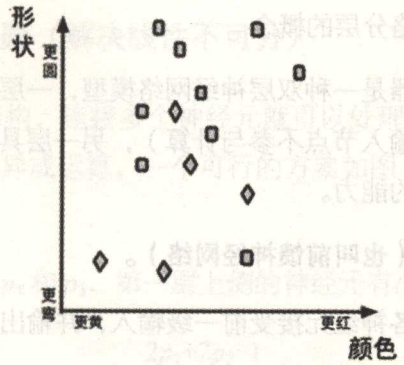


图 2-12 线性不可分

完全傻眼，无法用一条直线分类。

再来看看之前我们用程序实现的感知机无法解决的 XOR 问题，XOR 问题就是无法用一条直线区分两类事物，这就是线性不可分问题，如图 2-13 所示。最右边的图要想把浅色和深色分开需要两条线。

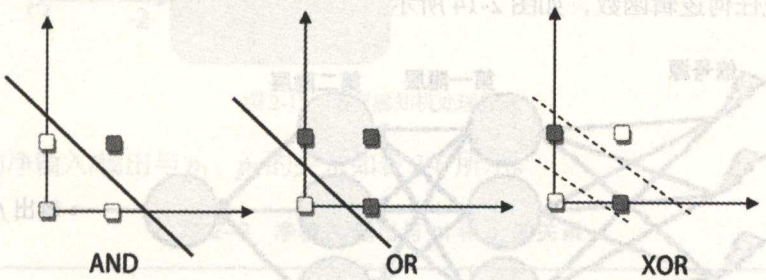


图 2-13 XOR 线性不可分

好的，在机器学习领域，有多种方法可以解决这类问题，本书讲的是神经网络和深度学习，所以我们需要用以下知识体系了解如何解决线性不可分问题。

1. 多层神经网络

还记得是谁提出的 XOR 解决方法吗？请回顾一下：1974 年，哈佛大学的一篇博士论文证明了在神经网络多加一层，并且利用“后向传播”（Back-propagation）学习方法，可以解决 XOR 问题。

我们先讲讲神经网络分层的概念。

（1）**感知器：**感知器是一种双层神经网络模型，一层为输入层（以上我们简化为输入刺激，也就是输入节点不参与计算），另一层具有计算单元，可以通过监督学习建立模式判别的能力。

（2）**多层神经网络（也叫前馈神经网络）。**

特点：前馈网络的各神经元接受前一级输入，并输出到下一级，无反馈。

节点：输入节点，输出节点。

（3）**计算单元：**可有任意一个输入，但只有一个输出，输出可耦合到任意多个其他节点的输入。

（4）**层：**可见层——输入和输出节点；隐层——中间层。

三层（一般我们只画出两层计算单元，输入层不参与计算不画出）前馈网络可以实现任何逻辑函数，如图 2-14 所示。

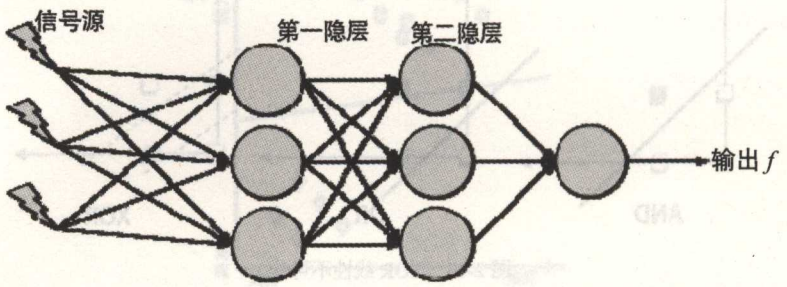


图 2-14　前馈神经网络

我们把中间单层神经元扩展成两个神经元层，并且赋予了新的名字。第一层神经元叫第一隐层，第二层神经元叫第二隐层，我们把由输入的信号源、隐层及输出组成的层叫作多层神经网络，多层神经网络里一个重要的特征是上一层输出只能是下一层输入，不可跨层链接。

2.5.2 解决 XOR 问题（解决线性不可分）

利用多层感知机结构，连接多个神经元就可以处理异或问题。使用一个两层神经网络就可以记忆异或运算。一个可行的方案如图 2-15 所示，f_1 使用 *Step* 函数。

网络接收两个输入 p_1 和 p_2，第一层上侧的神经元有净输入：

$$2p_1+2p_2-1$$

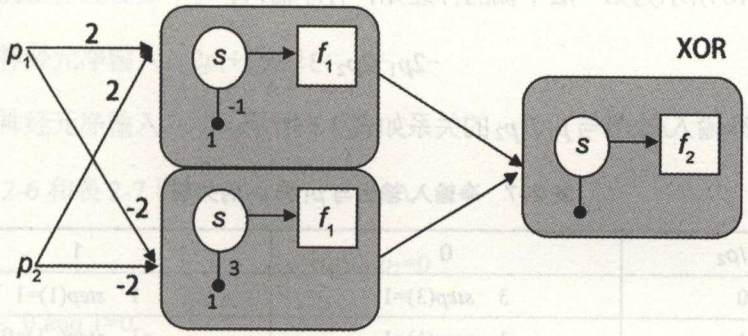


图 2-15 多层感知机处理异或

它的净输入/输出与 p_1、p_2 的关系如表 2-6 所示。

表 2-6 净输入/输出与 p_1 和 p_2 的关系

p_1/p_2	0	1
0	−1 *step*(−1)=0	1 *step*(1)=1
1	1 *step*(1)=1	3 *step*(3)=1

实际上，这个神经元对输出结果做了如图 2-16 所示的划分。

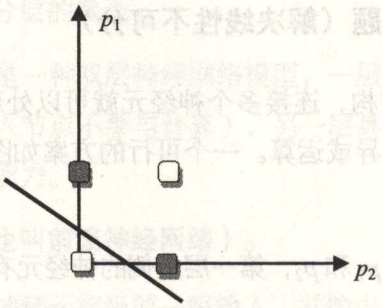


图 2-16　上侧神经元的划分

图 2-16 所示为第一层下侧的神经元，有净输入：

$$-2p_1-2p_2+3$$

它的净输入/输出与 p_1、p_2 的关系如表 2-7 所示。

表 2-7　净输入/输出与 p_1 和 p_2 的关系

p_1/p_2	0	1
0	3　$step(3)=1$	1　$step(1)=1$
1	1　$step(1)=1$	−1　$step(-1)=0$

因此，下侧神经元对数据进行了如图 2-17 所示的划分。

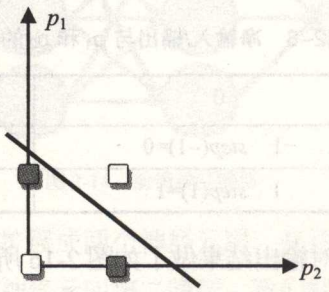


图 2-17　下侧神经元的划分

最后，由输出神经元对两个神经元的数据进行整合，这里使用逻辑与操作，得到图 2-18 所示的正确的异或运算的划分。

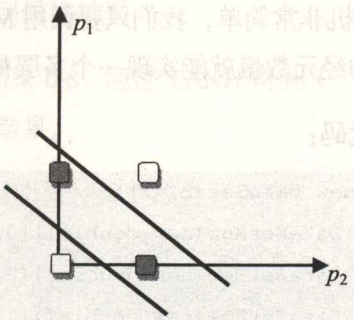


图 2-18 正确异或运算划分

具体的工作过程如下。

上侧神经元净输入：$2p_1+2p_2-1$

下侧神经元净输入：$-2p_1-2p_2+3$

查表 2-6 和表 2-7 可知：

$$p_1=0, p_2=0$$

输出：0 and 1=0

$$p_1=0, p_2=1$$

输出：1 and 1=1

$$p_1=1, p_2=0$$

输出：1 and 1=1

$$p_1=1, p_2=1$$

输出：1 and 0=0

使用多层感知机成功解决了异或问题。

2.5.3 XOR 问题的代码实现

我们来看看如何用 Neuroph 实现多层感知机，也就是多层神经网络。在

Neuroph 中使用多层感知机非常简单，我们只要调用 MultiLayerPerceptron 类，并且设定好层数及每层的神经元数量就能实现一个多层神经网络。

下面我们看看具体代码：

```
DataSet trainingSet = new DataSet(2, 1);
trainingSet.addRow(new DataSetRow(new double[]{0, 0}, new double[]{0}));
trainingSet.addRow(new DataSetRow(new double[]{0, 1}, new double[]{1}));
trainingSet.addRow(new DataSetRow(new double[]{1, 0}, new double[]{1}));
trainingSet.addRow(new DataSetRow(new double[]{1, 1}, new double[]{0}));
//创建多层感知机，输入层 2 个神经元，隐含层 3 个神经元，最后输出层为 1 个隐含神经元，我们使用 TANH 传输函数用于最后格式化的输出。
MultiLayerPerceptron myMlPerceptron = new
MultiLayerPerceptron(TransferFunctionType.TANH, 2, 3, 1);
      //开始训练
      System.out.println("Training neural network...");
      myMlPerceptron.learn(trainingSet);
```

看看输出结果：

```
.....
231. iteration : 0.010857327129718937
232. iteration : 0.010643221657202485
233. iteration : 0.010435288982776427
234. iteration : 0.010233311250628364
235. iteration : 0.01003707920734522
236. iteration : 0.009846391849799534
[main] INFO org.neuroph.core.learning.LearningRule - Learning Stoped
Testing trained neural network
Input: [0.0, 0.0] Output: [0.03794371028454834]
Input: [0.0, 1.0] Output: [0.8280251075783817]
Input: [1.0, 0.0] Output: [0.8061029856179274]
Input: [1.0, 1.0] Output: [0.0931520677705696]
```

经过 236 次计算，我们的程序学习完毕并输出结果，Input:[0.0,0.0]，输出 0.037

经过 TANH 函数处理，得到最终结果 0。

Input:[0.0, 1.0]，输出结果 0.8，经过 TANH 得到 1，依此类推，我们最后得到 XOR 异或学习的准确计算结果。

我们再来看看改变隐层的神经元个数会怎么样？

```
new MultiLayerPerceptron(TransferFunctionType.TANH, 2, 3, 1);
改成 new MultiLayerPerceptron(TransferFunctionType.TANH, 2, 2, 1);
```

计算结果：

```
.....
224. iteration : 0.010137332719035904
225. iteration : 0.009894060755430178
[main] INFO org.neuroph.core.learning.LearningRule - Learning Stoped
Testing trained neural network
Input: [0.0, 0.0] Output: [0.04957962733734304]
Input: [0.0, 1.0] Output: [0.8184942278091288]
Input: [1.0, 0.0] Output: [0.8006225511852936]
Input: [1.0, 1.0] Output: [0.04613571126703119]
```

没有太大变化，所有的变化都在正常的误差内。

如果我们将中间层变成 30，会发生什么变化呢？

```
2409. iteration : 0.002927935122956385
[main] INFO org.neuroph.core.learning.LearningRule - Learning Stoped
Testing trained neural network
Input: [0.0, 0.0] Output: [-0.022022339547133944]
Input: [0.0, 1.0] Output: [0.9341172653775799]
Input: [1.0, 0.0] Output: [0.9069350056674689]
Input: [1.0, 1.0] Output: [0.009379875622236349]
```

计算了 2409 次才完成，太耗时了，看看得到的数字，-0.022、0.93、0.90、0.009，更趋向于 1 和 0，换句话说就是更精确了。

好的，我们仅仅靠调整隐层的神经元个数就已经能优化一个较为精确的数字了。

2.6 解决一些实际问题

2.6.1 识别动物

设想我们去动物园观赏动物，想要把羚羊和龙虾区分开来是很容易的，因为它们的显著特征都不太一样，比如羚羊有 4 条腿，虾一般 6 对以上条腿。复杂一点点的动物，例如羚羊和野猪怎么区分呢？

任何神经网络的学习都是学习特征的一个过程，不管是人为定义还是机器自动学习的，这里我们人为定义动物的特征如下。

属性序号:属性:属性类型

1:是否有毛发:布尔型

2:是否有羽毛:布尔型

3:是否卵生:布尔型

4:是否哺乳:布尔型

5:是否可以飞行:布尔型

6:是否水生:布尔型

7:是否食肉:布尔型

8:是否有牙齿:布尔型

9:是否有脊椎:布尔型

10:是否呼吸空气:布尔型

11:是否有毒的:布尔型

12:是否有鳍:布尔型

13~17:腿:数字（设置的值：{0、2、4、6、8}，用二进制表示，比如2就用01000，4就用00100，6就用00010等）

18:是否有尾巴:布尔型

19:是否可驯养:布尔型

20:猫科:布尔型

根据以上特征，我们将动物归类为7种。

输出：

类型：数字［整数的值范围（1、7）］用二进制表示，7用0000001表示，6用0000010表示，依此类推

类——物的设置：

7——（41）土豚、羚羊、熊、野猪、水牛、小牛、豚鼠、猎豹、鹿、海豚、大象、果蝠、长颈鹿、母畜、山羊、大猩猩、仓鼠、兔子、豹子、狮子、猞猁、水貂、鼹鼠、猫鼬、负鼠、羚羊、鸭嘴兽、臭鼬、小马、海豚、美洲狮、猫咪、浣熊、驯鹿、海豹、海狮、松鼠、吸血蝙蝠、田鼠、袋鼠、狼

6——（20）鸡、乌鸦、鸽子、鸭、火烈鸟、海鸥、鹰、几维、云雀、鸵鸟、鹦鹉、企鹅、雉鸡、土卫五、燕鸥、贼鸥、麻雀、天鹅、秃鹫、鹪鹩

5——（5）响尾蛇、蛇、蛇晰、乌龟、蜥蜴

4——（13）鲈鱼、鲤鱼、鲶鱼、鲑鱼、角鲨、黑线鳕、鲱鱼、梭子鱼、食人鱼、海马、鳎目鱼、黄貂鱼、金枪鱼

3——（4）蛙、青蛙、蝾螈、蟾蜍

2——（8）跳蚤、蚊子、蜜蜂、家蝇、瓢虫、蛾、白蚁、黄蜂

1——（10）蛤蜊、螃蟹、小龙虾、龙虾、章鱼、蝎子、海黄蜂、蛞蝓、海星、蠕虫

以上，我们定义了一个简单的输入输出模型，输入是 20 个数字，分别代表各个特征，输出定义为 7 个数字，分别代表 7 类分类结果。

建立如图 2-19 所示的神经网络结构。

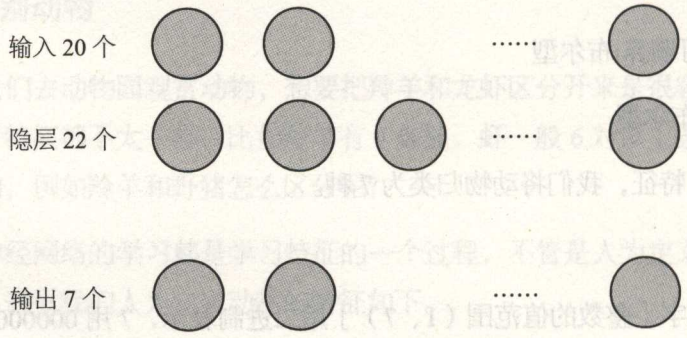


图 2-19　动物识别的神经网络结构

下面我们来看看程序是怎么实现的。

```
//定义 animals 训练数据
String trainingSetFileName = "data_sets/animals_data.txt";
//定义两个变量分别为输入层神经元个数 20，输出层神经元个数 7 个
      int inputsCount = 20;
      int outputsCount = 7;
      //取出并建立训练数据集
      DataSet dataSet = DataSet.createFromFile(trainingSetFileName,
inputsCount, outputsCount, "\t", true);
      System.out.println("Creating neural network...");
      //建立神经网络，定义一个隐层，神经元设为 22
      MultiLayerPerceptron neuralNet = new
MultiLayerPerceptron(inputsCount, 22, outputsCount);
      //定义学习规则、BP
      MomentumBackpropagation learningRule = (MomentumBackpropagation)
neuralNet.getLearningRule();
```

```
        learningRule.addListener(this);
        //设置最大误差、学习速度
        learningRule.setLearningRate(0.2);
        learningRule.setMaxError(0.01);
        System.out.println("Training network...");
        //开始训练
        neuralNet.learn(dataSet);

        System.out.println("Training completed.");
        System.out.println("Testing network...");
        testNeuralNetwork(neuralNet, dataSet);
```

运行程序，部分运行结果截取如下：

```
……
98. iteration | Total network error: 0.010074665914484453
99. iteration | Total network error: 0.010022261662764946
100. iteration | Total network error: 0.009971988745191207
[main] INFO org.neuroph.core.learning.LearningRule - Learning Stoped
100. iteration | Total network error: 0.009971988745191207
Training completed.
Testing network...
Input: [1.0, 0.0, 0.0, 1.0, 0.0,
0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0]
Output: [1.974908450012832E-6, 0.002271565834507692, 0.002853187779537703,
2.9595696050419264E-5, 6.595289445574359E-4, 3.7820430825616786E-5,
0.9948826549153703]
Input: [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 1.0, 0.0, 1.0] Output: [2.899283987949604E-7,
6.229717751888676E-4, 7.23163571074833E-4, 1.2120900294516567E-4,
0.005912830298777002, 2.441140187392956E-4, 0.9941521184117198]
Input: [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 1.0, 0.0, 0.0] Output: [0.010140140920781894,
```

```
3.6144138454713016E-7, 0.0014665024614726318, 0.9806509123590715,
0.01597342503248453, 0.001971567639749188, 0.13299844073676065]
Input: [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0] Output: [1.974908450012832E-6,
0.002271565834507692, 0.002853187779537703, 2.959569605041926 4E-5,
6.595289445574359E-4, 3.7820430825616786E-5, 0.9948826549153703]
……
```

先看我们训练出来的第一个结果集：

```
Input: [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0, 0.0, 1.0] Output: [1.974908450012832E-6,
0.002271565834507692, 0.002853187779537703, 2.959569605041926 4E-5,
6.595289445574359E-4, 3.7820430825616786E-5, 0.9948826549153703]
```

这组数字中 input 部分是 20 个数字，输入是：

有毛发，无羽毛，非卵生，哺乳，不能飞行，……根据这些特征，我们人工判别此动物为土豚，属于第 7 种类别的动物。

再看看输出：

```
Output: [1.974908450012832E-6, 0.002271565834507692, 0.002853187779537703,
2.9595696050419264E-5, 6.595289445574359E-4, 3.7820430825616786E-5,
0.9948826549153703]
```

其他位近似处理为 0，最后一位处理为 1，我们参照之前的分类，得到的输出对应的是第 7 种类别的动物，其中第一个就是土豚，和我们人工判别的动物类型完全一致！

从这个例子可以看出，输入层和输出层一般按照数据集和需求确定，隐层的神经元个数需大于输入层，具体个数可以调整，主要看需求是什么：精度高还是程序运行速度快？

2.6.2 我是预测大师

神经网络只能解决分类问题吗？当然不是说分类不重要，分类及特征识别非常重要，从大量的已知模式中学习东西非常重要。神经网络只能解决分类的问题吗？实际上，无论在学术界还是应用层面上，人们十分关心神经网络的预测能力！

举几个耳熟能详的例子。

1. 谷歌借助大数据预测流感的爆发趋势

“谷歌流感趋势”是谷歌 2008 年推出的用于预警流感的即时网络服务。该系统根据对流感相关关键词的搜索进行数据挖掘和分析，创建对应的流感图表和地图，目前可预测全球超过 25 个国家的流感趋势，如图 2-20 所示。

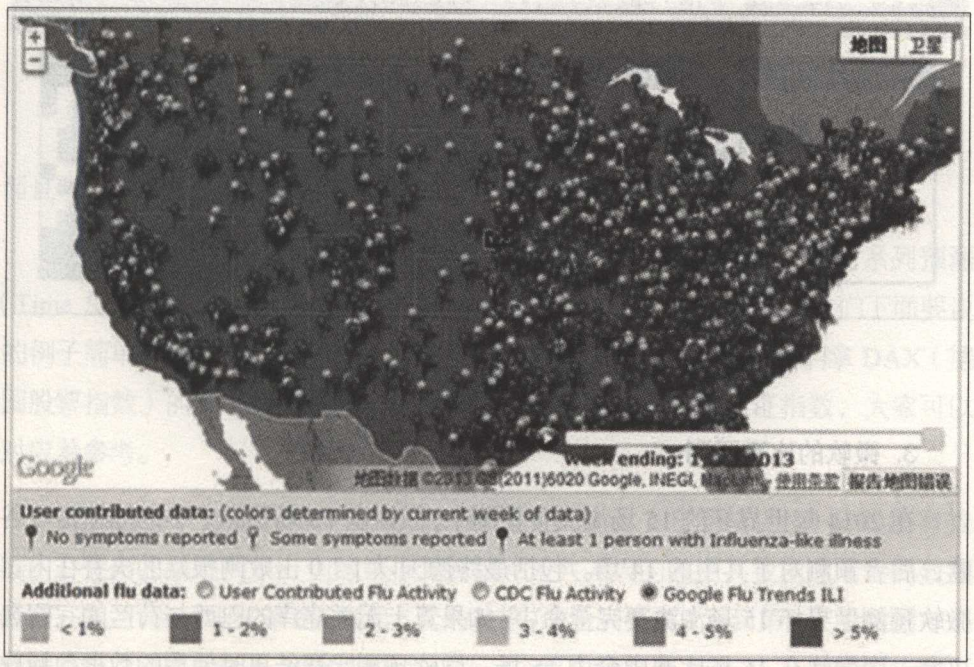


图 2-20 大数据预测流感爆发趋势

2. IBM 在孟菲斯市应用的预测分析技术

通过利用 IBM 预测分析软件预测趋势、分配资源并识别热点地区，田纳西州孟菲斯市部署了警察局的 Blue CRUSH（利用统计历史减少犯罪）方法。孟菲斯市用 IBM SPSS 预测分析软件改善警察局整体运行，显著降低了犯罪率，并在没有大量增加人员的情况下扩展了管辖区域的范围。孟菲斯警察局在短短 2.7 个月内就实现了 863%的投资回报，如图 2-21 所示。

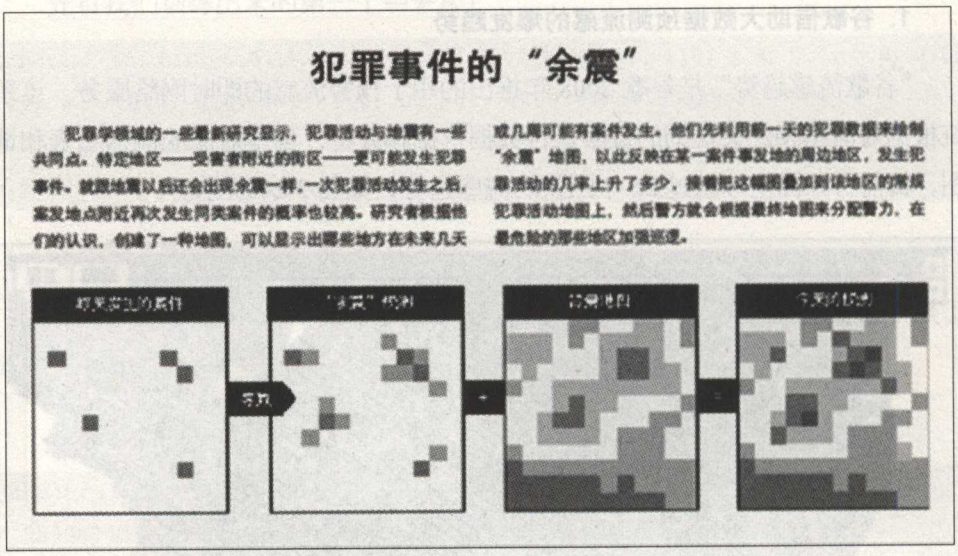


图 2-21　IBM 在孟菲斯市应用的预测分析

3. 微软的比赛预测

在 2014 年世界杯的 15 场淘汰赛预测中，微软以 15 场预测全中的战绩击败谷歌，后者预测对了其中的 14 场。包括最终德国以 1∶0 击败阿根廷的决赛在内，微软预测世界杯 15 场淘汰赛完全命中。如果算上无关痛痒的巴西与荷兰的三四名决赛，则微软在 16 场比赛中命中 15 场。微软预测比赛结果时使用的数据包括球队的胜负记录、赛事安排的强度、以往比赛的胜率、主场优势、天气等因素，如图 2-22 所示。

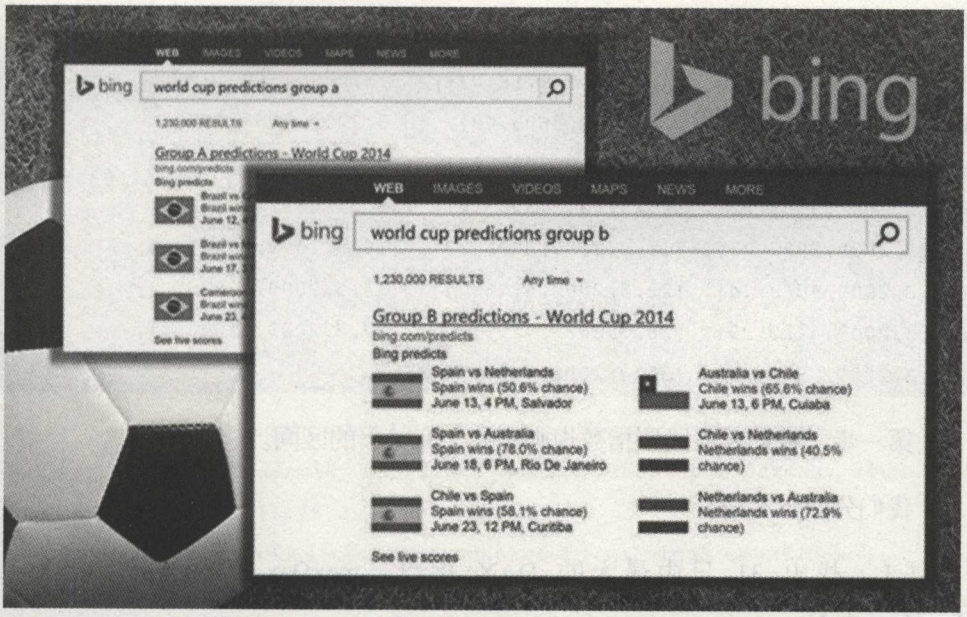


图 2-22 微软的比赛预测

我说这些的目的是什么呢？我想告诉大家，这些预测案例神经网络也能做到，而且可以做得更好！下面我们来看一个股票预测的例子。

从预测的角度看，实际上所有的预测都是对时间线的预测，时间系列预测（Time Series Prediction）在经济领域扮演了一个非常重要的角色，我们下面要说的例子简单介绍了如何将经济领域的指数预测变成可能。首先，我们拿 DAX（德国股票指数）的数字作为输入，这个数字非常接近写书时的上证指数，大家可以对应着参考。

我们选择了 DAX 2009 年 3 月的数据（从 3 月 2 日到 3 月 30 日的所有数据）。主要思路很简单：训练第 1 天到第 4 天共 4 天的数据，找出这 4 天的数据特征并学习，然后训练第 2 天到第 5 天的数据，然后训练第 3 天到第 7 天的数据……依此类推，训练完所有 30 天的内容，让这个算法计算出 3 月 31 日的股票指数，最后可以比对预测的数据和实际数据差异。

我们来看看如何实现：所有的数据是包含一个时间线的股票指数数据，用双

精度字符表示。

```
double[ ][ ] days = {{2,3,2009,3710.07}, {3,3,2009,3690.72},
{4,3,2009,3890.94}, {5,3,2009,3695.49}, {6,3,2009,3666.41},
{9,3,2009,3692.03}, {10,3,2009,3886.98}, {11,3,2009,3914.1},
{12,3,2009,3956.22}, {13,3,2009,3953.6}, {16,3,2009,4044.54},
{17,3,2009,3987.77}, {18,3,2009,3996.32}, {19,3,2009,4043.46},
{20,3,2009,4068.74}, {23,3,2009,4176.37}, {24,3,2009,4187.36},
{25,3,2009,4223.29}, {26,3,2009,4259.37}, {27,3,2009,4203.55},
{30,3,2009,3989.23}, {31,3,2009,4084.76}};
```

第一步，我们要将股票指数规则化到（0，1）的区间。

我们分两步完成。

（1）找出 31 日中最大的 DAX 指数，maxDax = max(days[k], k=0, days.length-1)。

（2）规则化指数 daxnorm [i] = (days [i] [3] / maxDax) × 0.8+0.1。这个函数将 0 ~ 10000 的自然数简单地规则化为从最小 0.0001 到最大 0.9999 的（0，1）区间的小数。

第二步，我们将确定用什么样的网络、网络具备多少层、每层有多少个神经元等问题。这些问题没有规律可循，每种预测可能都不一样。

给出一些经验值，首先，对于这个预测，单个感知机或者单层感知机都无法胜任，多层感知机是必备的，起码具备一个隐层，但隐层的神经元数量是多少呢？这里有个大致的推理公式：$2n+1$，n 是输入单元的个数。

在这个案例中，因为我们需要输出的是指数，而且只有一个指数，所以输出层只包含一个神经元。

我们设定一些基本信息 maxIteration=10000，learningRate=0.7，maxerror=0.0001 就开始启动程序吧。

```
    int maxIterations = 10000;
```

```
        NeuralNetwork neuralNet = new MultiLayerPerceptron(4, 9, 1);
        ((LMS) neuralNet.getLearningRule()).setMaxError(0.001);//0-1
        ((LMS) neuralNet.getLearningRule()).setLearningRate(0.7);//0-1
        ((LMS) neuralNet.getLearningRule()).setMaxIterations(maxIterations);//0-1
        TrainingSet trainingSet = new TrainingSet();

        double daxmax = 10000.0D;
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3710.0D / daxmax, 3690.0D / daxmax, 3890.0D / daxmax, 3695.0D /
daxmax}, new double[]{3666.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3690.0D / daxmax, 3890.0D / daxmax, 3695.0D / daxmax, 3666.0D /
daxmax}, new double[]{3692.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3890.0D / daxmax, 3695.0D / daxmax, 3666.0D / daxmax, 3692.0D /
daxmax}, new double[]{3886.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3695.0D / daxmax, 3666.0D / daxmax, 3692.0D / daxmax, 3886.0D /
daxmax}, new double[]{3914.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3666.0D / daxmax, 3692.0D / daxmax, 3886.0D / daxmax, 3914.0D /
daxmax}, new double[]{3956.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3692.0D / daxmax, 3886.0D / daxmax, 3914.0D / daxmax, 3956.0D /
daxmax}, new double[]{3953.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3886.0D / daxmax, 3914.0D / daxmax, 3956.0D / daxmax, 3953.0D /
daxmax}, new double[]{4044.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3914.0D / daxmax, 3956.0D / daxmax, 3953.0D / daxmax, 4044.0D /
daxmax}, new double[]{3987.0D / daxmax}));
```

```
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3956.0D / daxmax, 3953.0D / daxmax, 4044.0D / daxmax, 3987.0D /
daxmax}, new double[]{3996.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3953.0D / daxmax, 4044.0D / daxmax, 3987.0D / daxmax, 3996.0D /
daxmax}, new double[]{4043.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{4044.0D / daxmax, 3987.0D / daxmax, 3996.0D / daxmax, 4043.0D /
daxmax}, new double[]{4068.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3987.0D / daxmax, 3996.0D / daxmax, 4043.0D / daxmax, 4068.0D /
daxmax}, new double[]{4176.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{3996.0D / daxmax, 4043.0D / daxmax, 4068.0D / daxmax, 4176.0D /
daxmax}, new double[]{4187.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{4043.0D / daxmax, 4068.0D / daxmax, 4176.0D / daxmax, 4187.0D /
daxmax}, new double[]{4223.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{4068.0D / daxmax, 4176.0D / daxmax, 4187.0D / daxmax, 4223.0D /
daxmax}, new double[]{4259.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{4176.0D / daxmax, 4187.0D / daxmax, 4223.0D / daxmax, 4259.0D /
daxmax}, new double[]{4203.0D / daxmax}));
        trainingSet.addElement(new SupervisedTrainingElement(new
double[]{4187.0D / daxmax, 4223.0D / daxmax, 4259.0D / daxmax, 4203.0D /
daxmax}, new double[]{3989.0D / daxmax}));
        neuralNet.learnInSameThread(trainingSet);
        System.out.println("Time stamp N2:" + new SimpleDateFormat("dd-
MMM-yyyy HH:mm:ss:MM").format(new Date()));

        TrainingSet testSet = new TrainingSet();
```

```
        testSet.addElement(new TrainingElement(new double[]{4223.0D /
daxmax, 4259.0D / daxmax, 4203.0D / daxmax, 3989.0D / daxmax}));

        for (TrainingElement testElement : testSet.trainingElements()) {
            neuralNet.setInput(testElement.getInput());
            neuralNet.calculate();
            Vector<Double> networkOutput = neuralNet.getOutput();
            System.out.print("Input: " + testElement.getInput());
            System.out.println(" Output: " + networkOutput);
        }
```

训练完模型后，我们使用 3 月 27 日～3 月 30 日的指数作为输入，预测 3 月 31 日股票指数，程序输出如下：

```
Input: [0.4223, 0.4259, 0.4203, 0.3989] Output: [0.4093802943765426]
```

得到的预测股票指数为 4093.80。

我们看到，该模型的输出值是 4093，而实际上 3 月 31 日的股票指数是 4084.76，相差了 9 个点左右，误差在 0.22%，说明该算法基本是可用的。

然后，我们继续使用这个模型预测了 4 次。

```
Input: [0.4223, 0.4259, 0.4203, 0.3989] Output: [0.41018667907630685]
Input: [0.4223, 0.4259, 0.4203, 0.3989] Output: [0.41066248816065887]
Input: [0.4223, 0.4259, 0.4203, 0.3989] Output: [0.4096531596715262]
Input: [0.4223, 0.4259, 0.4203, 0.3989] Output: [0.4109325343858132]
```

预测的股票指数分别是 4101、4106、4096 和 4109，与实际值 4084 的误差在 9～15 个点左右，也就是 0.22%～0.3%，对于一个简单的 MLP 网络来说，我们觉得这个误差值是可以接受的。如果大家有兴趣，可以尝试着增加输入特征、增加神经网络层数等，尽量减少实际误差值。

3 深度学习是个什么东西

学习深度学习之前，首先要明白一些概念，我先把这些概念抛出来，然后逐个解释。

1. 机器学习

2. 人工神经网络

3. 深度神经网络

4. 深度学习

先来个维基百科的解释。深度神经网络是一种具备至少一个隐层的神经网络。与浅层神经网络类似，深度神经网络也能为复杂非线性系统提供建模，但多出的层次为模型提供了更高的抽象层次，因而提高了模型的能力。深度神经网络通常都是前馈神经网络，但也有语言建模等方面的研究将其拓展到循环（递归）神经网络。卷积深度神经网络（Convolutional Neuron Networks，CNN）在计算机视觉领域得到了成功的应用。此后，卷积神经网络也作为听觉模型被使用在自动语音识别领域，较以往的方法获得了更优的结果。

从这段解释我们可以发现如下两点。

（1）深度神经网络通常是前馈型神经网络，什么是前馈型神经网络，如果你仔细看书的话，应该已经明白了：它是神经网络的一种构型，特征是至少有一个隐层。反过来说，多层前馈型神经网络都是深度神经网络，但深度神经网络不都是多层前馈型神经网络，因为深度神经网络还包含了递归神经网络和卷积神经网络。卷积神经网络很重要，我们放在后面讨论。

（2）深度神经网络说的是一种结构，而不是一种算法！

3.1 机器学习

第 0 章中我们讲了很多关于机器学习的东西，机器学习是人工智能的一个分支，它本身是非常大的概念，本书提到的所有算法和理论都在机器学习这个概念之下。

什么是机器学习？以下是两种解释。

（1）机器学习算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法。

（2）机器学习就是通过算法，使得机器能从大量历史数据中学习规律，从而对新的样本做智能识别或对未来做预测。

我个人喜欢第二种解释，显得接地气。

机器学习的概念很大，我们先按照学习方式分类，试着尽量说明白。

1．监督学习（Supervised Learning）

监督学习是机器学习中一种典型的学习方法，顾名思义，监督式学习就是有个人在旁边看着你学习，随时纠正你学习中的错误。怎么纠正呢？对错误学习给予惩罚，对正确学习给予奖励。

如图 3-1 所示，监督式学习中，所有输入数据都被称为“训练数据”，每组

数据其实由两个部分构成：正确的训练集和错误的训练集，这两部分训练集最好在数量上相等，并且样本数量足够全，能覆盖到最多的情况。接下来，将训练这些数据并将识别或预测结果与实际的人为标定过的“训练数据”做比较，不断地调整模型，直到模型的预测结果达到一个预测的准确率。

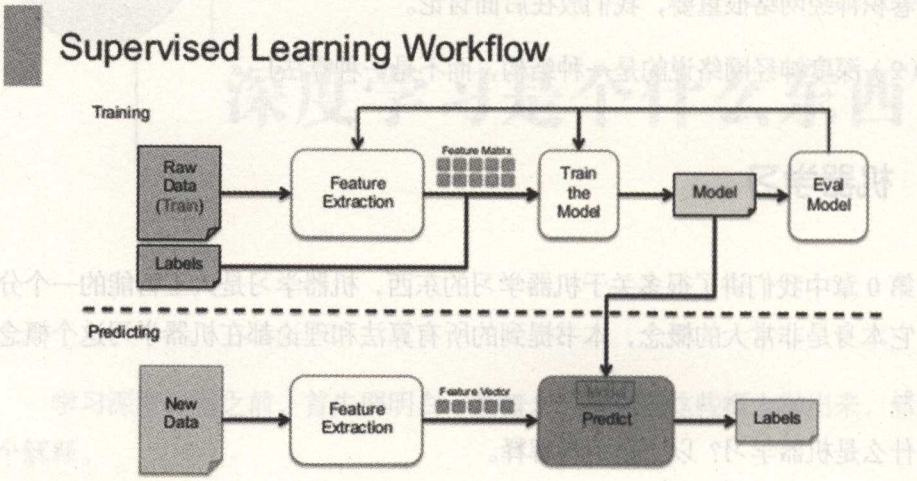


图 3-1　监督学习

2. 非监督学习（Unsupervised Learning）

如图 3-2 所示，非监督学习，也就是没有“人”在旁边督促你学习，没有人为标定好的训练数据，没有告诉模型哪些数据是正确的，哪些是不正确的。在非监督式学习中，学习模型是为了推断出数据的一些内在结构。很常见的应用场景包括关联规则的学习及聚类等。比如说，我们将苹果和香蕉混合在一起，并没有告诉模型苹果和香蕉的特征，模型自己去做聚类学习，有可能学习出来很多种类别，除了特征相差较大的苹果和香蕉两种不同的水果外，还能发现某些苹果和香蕉的特殊品种，这种发现是由算法自己找出的。非监督式学习中的常见算法包括 Apriori 算法及 K-Means 算法。

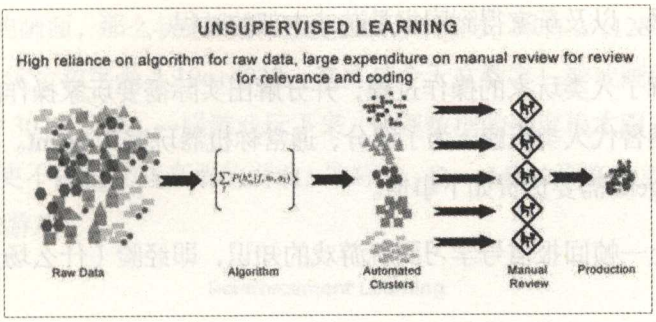


图 3-2 非监督学习

3. 强化学习（Reinforcement Learning，RL）

要了解强化学习，我们要先看清楚人类是怎么玩游戏的。

首先，游戏开始时停留在初始时刻。然后，游戏场景开始改变，玩家眼睛捕捉到场景的变化，将视觉信号传回大脑皮层处理。

之后，大脑皮层将视觉信号转换为人类理解的含义，通过之前玩游戏的经验，将理解的含义与应该进行的操作做映射，之后将映射后得到的操作信号传递到身体并产生一个动作，如手指动作。操作结束后，游戏场景进入下一帧，玩家得到回报，如越过关口，或者吃到金币。如此循环，直到游戏结束，如图 3-3 所示。

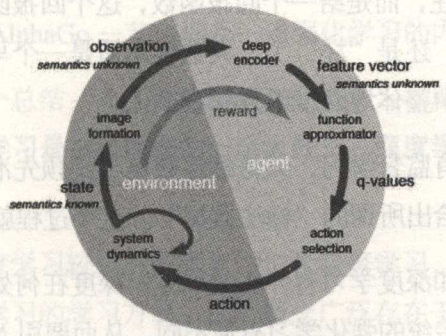


图 3-3 游戏循环流程图

仔细想想这个过程，发生在游戏内部的那些事情不用玩家考虑，玩家能够覆盖的只是上述游戏循环的右半段，即输入视觉信号，输出手指动作。从手指动作

到下一帧场景，以及玩家得到回报是游戏内部的过程。

既然了解了人类玩家的操作过程，并分解出实际需要玩家操作的内容，下一步就是让机器替代人类玩家。为了区分，通常称机器玩家为 agent。与人类玩家的操作类似，agent 需要负责如下事情。

（1）由上一帧回报信号学习到玩游戏的知识，即经验（什么场景下需要什么操作）。

（2）视觉信号的处理与理解（降维，高层特征抽取）。

（3）根据经验及高层的视觉特征，选择合理的经验（动作）。

（4）从动作反馈到游戏，即玩家手动的部分。

所以说，游戏都是越玩越好的，人类玩家如此，agent 亦如此。既然已经刻画出操作步骤，随着深度学习和强化学习的发展，实操也不是什么难题。下面，我们先看看强化学习是如何促进 agent 学习的。

强化学习其实就是一个连续决策的过程，如图 3-4 所示。传统的机器学习中的有监督学习是给定一些标注数据，学习一个好的函数，对未知数据做出很好的决策。但是有时候，你不知道标注是什么，即一开始不知道什么是“好”的结果，所以 RL 不是给定标注，而是给一个回报函数，这个回报函数决定当前状态得到什么样的结果（“好”还是“坏”），其数学本质是一个马尔科夫决策过程。最终的目的是决策过程中整体地回报函数期望最优。

这个过程有点像有监督学习，只是标注数据不是预先准备好的，而是通过一个过程来回地调整并给出所谓的“标注数据”。这个过程就是强化学习。

那么，强化学习和深度学习有什么关系呢？深度在何处？换句话说，深度学习参与的强化学习与传统的强化学习有何不同，从而要引入深度学习呢？

我们在前面介绍强化学习的过程中，处理的是状态。实际上，很多时候状态是连续的、复杂的、高维的。不像之前介绍中说的 4 个状态就可以了。假设我们

有 128 × 128 的画面，那么状态的数目是指数级增长的，即有 2^(128 × 128)种可能存在的状态，这个数字是 1.19e+4932，这可是个天文数字！游戏画面连续存在，就算按照每秒 30 帧来算，一局游戏玩下来，处理数据的速度根本跟不上游戏画面变化的速度，更不用说那些高清的游戏。实际上，DeepMind 现在也就能玩玩 Atari 这种爸爸辈的游戏。

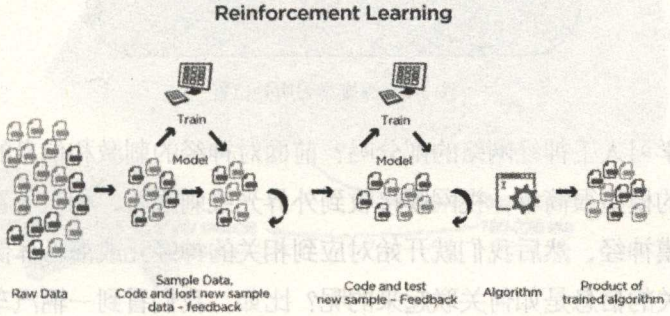


图 3-4　强化学习

无奈，因此求助于深度学习。注意，在此之前有很多人工特征处理，但很明显，一旦引入了人类的活动，就无法做成一种集成性的系统，只能成为实验室的二维画面玩具。人类为什么玩游戏玩得好呢？因为人脑非常善于处理高维数据，并飞快地从中抽取模式。现在由深度学习来替代这块短板。

我们在本书中的 AlphaGo 一节会涉及一点强化学习的内容。

现在我们来做一个总结：人工神经网络和深度学习都属于机器学习的一种，如图 3-5 所示。**深度学习是神经网络的一个大分支，深度学习的基本结构是深度神经网络。**

本书我们介绍深度学习较多，深度学习包含监督学习、非监督学习等，但监督和非监督是指机器学习的学习方式，这个概念广泛存在于机器学习中，而不只有深度学习里有这两个概念，这个概念需要请读者注意。

深度学习的地位搞清楚了，我们再来研究下大脑，仍然回到大脑是如何工作的问题上。

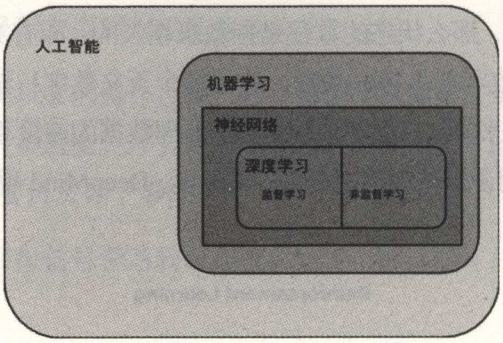


图 3-5　深度学习所处位置

还记得学习人工神经网络的部分吗？前面对神经的刺激和信息的存储都有过讨论，当时的模型很简单，视网膜接收到外界光线刺激后，相当于看到图像后，传递给视网膜神经，然后我们就开始对应到相关的神经元或海马体储存下来，那么图像和相关的信息是如何关联起来的呢？比如，我们看到一辆汽车，如何将汽车这副图像和脑中相关的概念联系起来呢？只有联系起来，我们才能有基本的联想或推理，比如，我看到这辆车了，这辆车看起来比较像一艘船，这就将船和汽车联系在一起；或者根据这辆车上有泥巴，判断这辆车昨天肯定去过土路，这是结合现实世界的一种简单推理。

不管是联想还是推理，我们都离不开识别物体，而大脑是如何做到的呢？

1981 年的诺贝尔医学奖，颁发给了 David Hubel、Torsten Wiesel 和 Roger Sperry。前两位的主要贡献是，发现了人的视觉系统的信息处理是分级的。如图 3-6 所示，从视网膜（Retina）出发，经过低级的 V1 区提取边缘特征，到 V2 区的基本形状或目标的局部，再到高层的整个目标（如判定为一张人脸），以及到更高层的 PFC（前额叶皮层）进行分类判断等。也就是说，高层的特征是低层特征的组合，从低层到高层的特征表达越来越抽象和概念化，即越来越能表现语义或者意图。

先注意第一句话，视觉系统的信息处理是分级的。这就告诉我们，人的视觉不是直接出现一辆车就识别一辆车，需要分级处理；如何分级处理？经过低级的 V1 区提取边缘特征，到 V2 区的基本形状或目标的局部，再到高层的整个目标（如判定为一辆车）。

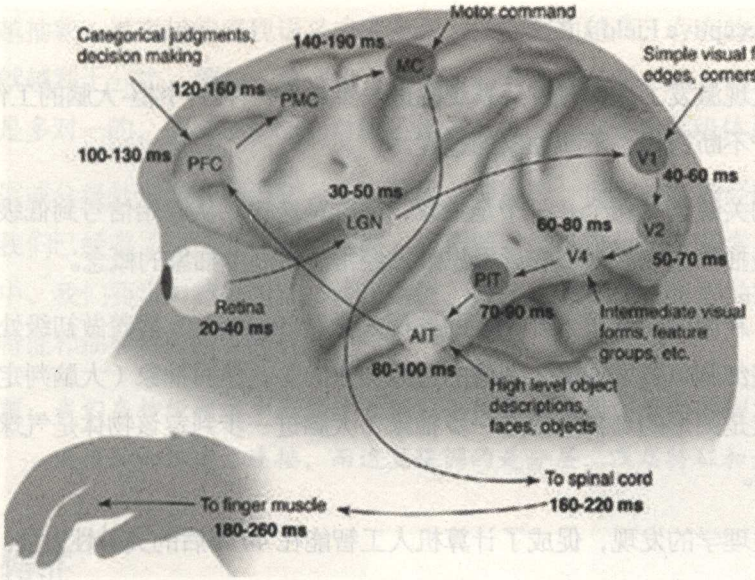


图 3-6 人脑结构图

看看“残忍”的科学家做了什么实验发现了这个现象。

1958 年，大卫·休伯尔（David Hunter Hubel）和托斯坦·维厄瑟尔（Torsten Wiesel）在约翰·霍普金斯大学（The JohnHopkins University）研究瞳孔区域与大脑皮层神经元的对应关系。他们在猫的后脑头骨上，开了一个 3 毫米的小洞，向洞里插入电极，测量神经元的活跃程度。

然后，他们在小猫的眼前，展现各种形状、各种亮度的物体。并且，在展现每一件物体时，还改变物体放置的位置和角度。他们期望通过这个办法，让小猫的瞳孔感受不同类型、不同强弱的刺激。

他们之所以做这个试验，目的是证明一个猜测。位于后脑皮层的不同视觉神经元，与瞳孔所受刺激之间，存在某种对应关系。一旦瞳孔受到某种刺激，后脑皮层的某部分神经元就会活跃。经历了很多天枯燥的试验，同时牺牲了若干只可怜的小猫，大卫·休伯尔和托斯坦·维厄瑟尔发现了一种被称为“方向选择性细胞（Orientation Selective Cell）”的神经元细胞。当瞳孔发现了眼前的物体的边缘，而且这个边缘指向某个方向时，这种神经元细胞就会活跃。就此，他们提出了感

受区域（Receptive Field）的概念。

这个发现激发了人们对神经系统的进一步思考。神经-中枢-大脑的工作过程，或许是一个不断迭代、不断抽象的过程。

这里的关键词有两个，一个是抽象，一个是迭代。从原始信号到低级抽象，逐渐向高级抽象迭代。人类的逻辑思维，经常使用高度抽象的概念。

例如，从原始信号摄入开始（瞳孔摄入像素 Pixels），接着做初级处理（大脑皮层某些细胞负责发现摄入图像的边缘和方向），然后抽象（大脑判定眼前的物体的形状是圆形的），然后进一步抽象（大脑进一步判定该物体是气球），如图 3-7 所示。

这个生理学的发现，促成了计算机人工智能在 40 年后的突破性发展。

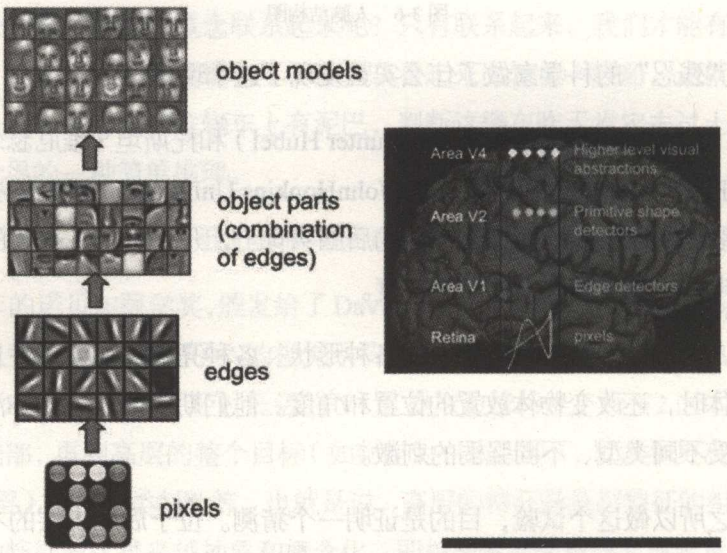


图 3-7　人脑各区域作用

再回顾下这个结论，人的视觉系统的信息处理是分级的。从低级的 V1 区提取边缘特征，到 V2 区形状或者目标的部分结构等，再到更高层，整个目标、目标的行为等。也就是说，高层的特征是低层特征的组合，从低层到高层的特征表

示越来越抽象，越来越能表现语义或者意图。抽象层面越高，存在的可能猜测就越少，就越利于分类。例如，单词集合和句子的对应是多对一的，句子和语义的对应也是多对一的，语义和意图的对应还是多对一的，这是个层级体系。

分层或分级处理是大脑识别一个物体的主要过程，而之前在神经网络的学习过程中我们已经有了隐层的概念，现在，我们要对特征进行研究和学习，在后面的章节中，我们要学习特征表达、浅层特征，以及机器如何将特征抽象出来，学习层级特征和抽象特征的不同点。

注意： 我们在神经网络中研究过大脑，但侧重点不同，当时的侧重点主要在连接和如何建立连接，而这里强调的是分层、浅层特征和如何抽象。

3.2 特征

特征是机器学习的原材料，上面我们做的水果识别的例子就是选取了水果的两个特征——颜色和形状。在什么层面上抽取特征和特征的精度决定了我们对各种水果的识别度，比如之前我们假定颜色的输入特征只有两种——红色和黄色，形状特征只有两种——圆形和弯形。这么简单的特征只能分类香蕉和苹果，而且还不精确，若我们遇到一个黄红色的苹果呢？或一根非常直的香蕉呢？如此粗略地选取特征无法生成一个理想的模型应用于现实生活中。必须要有新的特征引入才可能会做出精准的识别和分类动作，比如，引入香蕉和苹果的触感、味道等。那么，如何选取特征的范围，更为精确地找到必要的特征呢？

3.2.1 特征粒度

我们的算法在一个什么粒度上表示特征，才能发挥作用？以一个图片为例，我们将特征提取得非常细，细到最小粒度像素级的特征，但这毫无价值。如图 3-8 所示，摩托车以像素级别为特征，根本得不到任何信息，无法进行摩托车和非摩托车的区分；而如果特征是一个结构性（或者说有含义）的特征，比如是否具有

车把手（handle），是否具有车轮（wheel），就很容易区分摩托车和非摩托车，学习算法才能发挥作用。

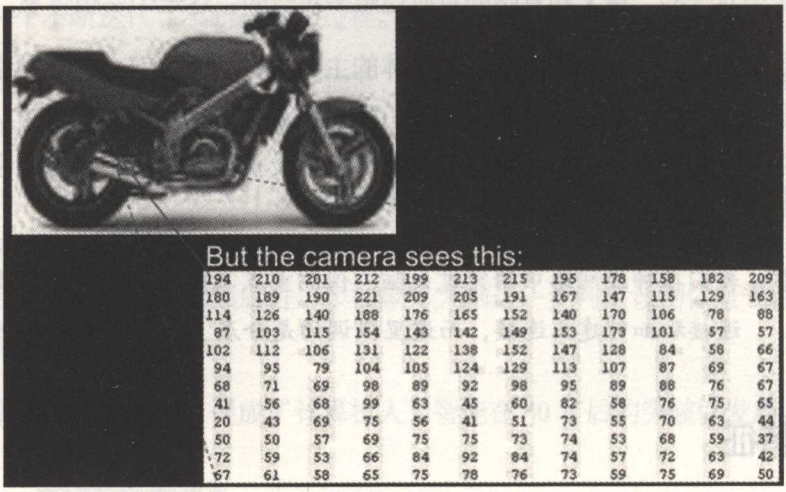


图 3-8　摄像头看到的摩托车

如图 3-9 所示，只有将像素级的组合抽象到一个程度，让机器学习车把手和车轮子的区别，才能识别出摩托车。

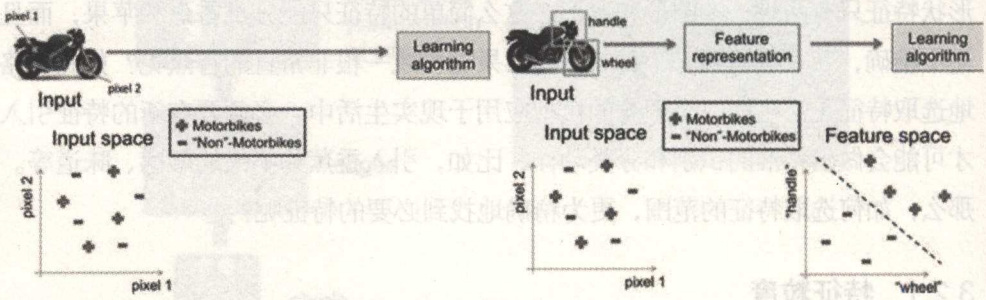


图 3-9　区分把手和轮子

3.2.2　提取浅层特征

如果像素级的特征没有任何用处，那我们如何将像素级的特征变为有一定识别度的浅层特征呢？

所有公开资料中，我能找到的信息如下。

1996 年，布鲁诺·奥尔斯豪森（Bruno Olshausen）和大卫·菲尔德（David Field）两位学者任职于康奈尔大学（Cornell University），他们在一定程度上解决了最浅层特征的问题。

他们收集了很多黑白的风景照片，首先，他们把每张照片都按 16×16 的像素规定一个方格，一共在照片上抽取 400 个这样的方格，这些方格的一部分是重叠的。然后随机抽取一张照片，从其中抽取一个方格，这个方格也是 16×16 像素的，记为 T，目的是从之前抽取的 400 个方格中找到一个方格无限接近于 T 的，尽可能和 T 相似。反复迭代后，得到了可以表示 T 的最佳的方格。他们发现，这些基本方格是不同物体不同方向的边缘线。

换句话说，再复杂的图形都是由基本图形构件组成，而且这种规律甚至可以推导到音频识别领域。

他们从未标注的声音中发现了 20 种基本的声音结构，其余的声音可以由这 20 种基本结构合成，如图 3-10 和图 3-11 所示。

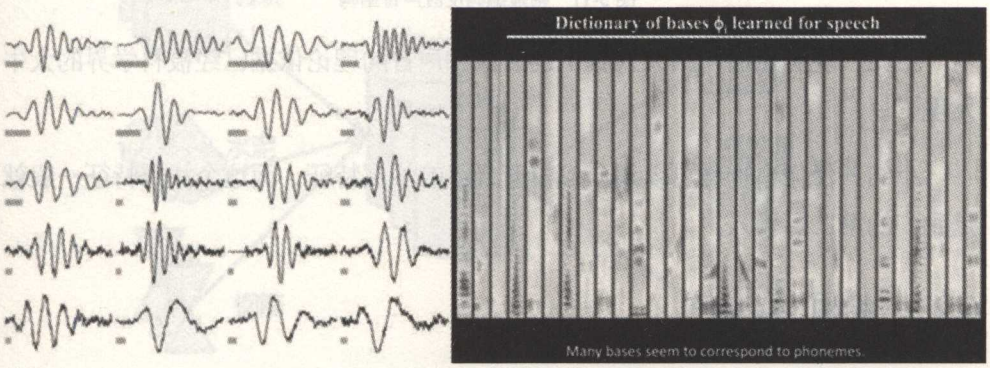


图 3-10 声音基本结构

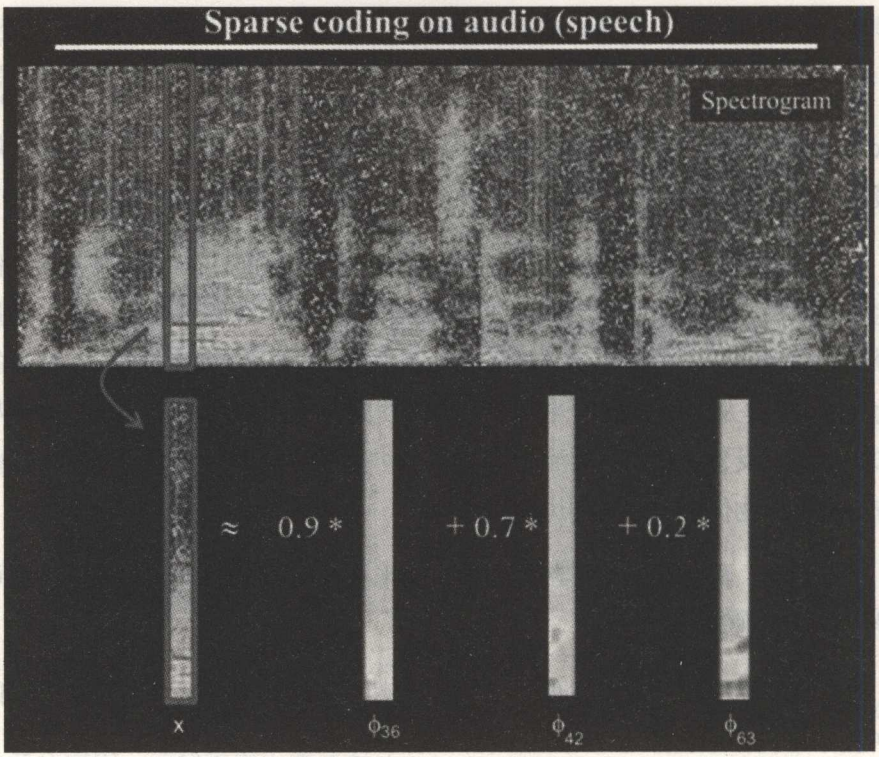


图 3-11　稀疏编码反应声音结构

至此，现实生活中两大人脑识别图像和声音的理论依据已经被科学界的大牛发现了。

小结：任何事物都可以划分成粒度合适的浅层特征，而这个浅层特征一般就是我们的第二层输入。

3.2.3　结构性特征

小块的图形可以由基本方格构成，那么，更复杂的，具有概念性的图形如何表示呢？这就需要更高层次的特征表示，比如 V2 和 V4。因此，V1 看输入层是像素级。V2 看 V1 层是像素级，这是层次递进的，高层表达由底层表达组合而成。专业点说就是基本构件。V1 层提出的基本构件是边缘，V2 层是 V1 层这些构件的组合，这时 V2 取得的又是高一层构件，即上一层的构件组合的结果，上上层

又是上一层的组合构件，如图 3-12 所示。

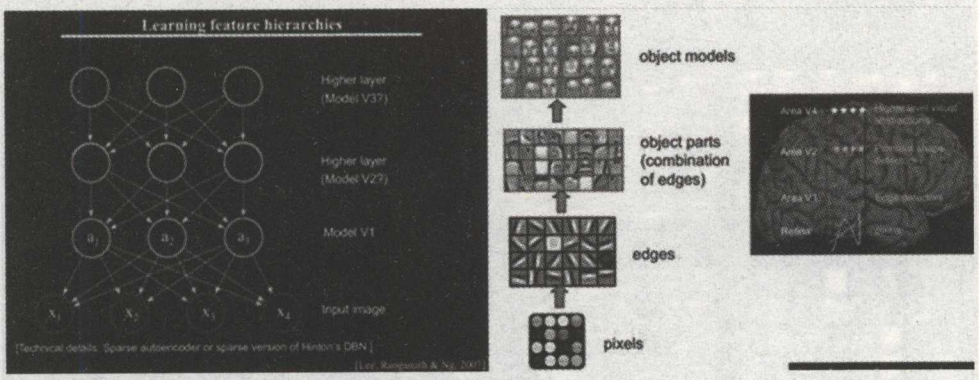


图 3-12 人脑和神经网络对比

我们用现实生活中盖房子的方法来解释这个概念，例如，我们要建设一个社区，这个社区由不同的功能建筑组成，如医院、学校、办公楼、住宅等。不管多复杂的建筑，构成的基本单位可以认为是每个混凝土结构，每个混凝土结构都由一些砖块、水泥和钢筋组成，如图 3-13 所示。

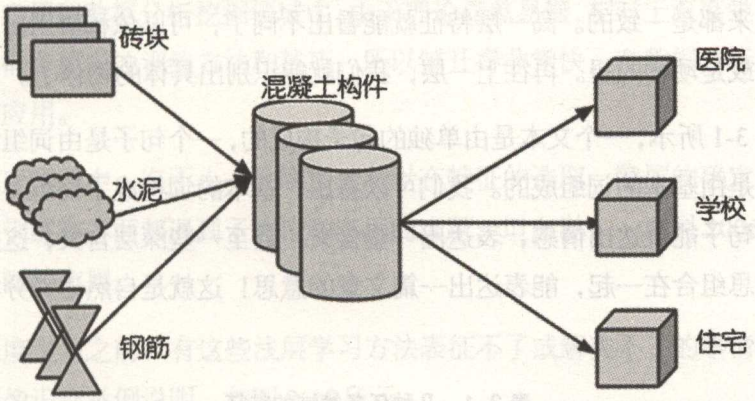


图 3-13 建筑构成

同样地，不管多复杂的图像，我们可以看到都是由基本的图像构件构成，这就像不管多复杂的建筑都是由水泥、砖块和混凝土组成。这些砖块、水泥、钢筋就是浅层特征，混凝土构件就是高一层级的特征，如图 3-14 所示。

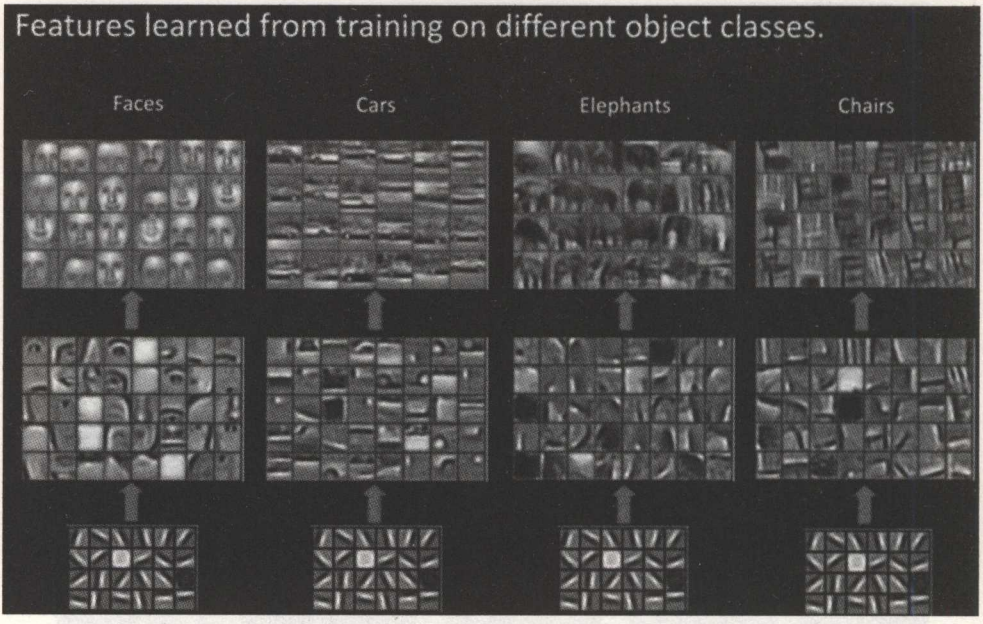


图 3-14　学习不同的图像，基本特征一致

现实生活中完全不一样的图片，可以由基本的组成单元构成，在底部所有的线条看起来都是一致的。高一层特征就能看出不同了，可以依稀辨别出人眼或汽车轮胎，或是动物的腿。再往上一层，我们就能识别出具体的物体了。

如表 3-1 所示，一个文本是由单独的句子构成的，一个句子是由词组组成的，一个词组是由基本的词组成的。我们可以看出，基本的词构成了词组，词组构成了句子，句子能表达出情感，表达出一些含义，甚至一些深层含义，这些句子表达出的意思组合在一起，能表达出一篇文章的意思！这就是自然语义分析的基本处理流程。

表 3-1　几种任务领域的特征

任务领域	原始输入	浅层特征	中层特征	高层特征	训练目标
语音	样本	频段	声音	音调	音素	单词	语音识别
图像	像素	线条	纹理	图案	局部	物体	图像识别
文本	字母	单词	词组	短语	句子	段落	文章	语义理解

小结：结构性特征具有明显的层级概念，从较小粒度划分，再用划分的基本特征组成上层特征，依此类推，可以展现特征的结构性。我们从图 3-14 可以看出，这就像一个多层神经网络表示每层次的递进关系。

3.3 浅层学习和深度学习

讲完了深度学习的一些概念，读者可能会问，有对应的“浅度学习”吗？在之前神经网络的学习中，含有一个隐层的就叫多层感知机，也叫神经网络，这种模型叫浅层模型。在这种模型中，人们可以用强于人工规则的统计学习方法，利用这种浅层模型实现较复杂的训练，甚至能实现对未知事件的预测，实现基本的分类。

到了 20 世纪 90 年代，各种各样的浅层机器学习模型被提出，例如，支持向量机（Support Vector Machines，SVM）、Boosting、最大熵方法（如 Logistic Regression，LR），这些模型都是只有一层或连一层隐层都没有的，提出后，它们很快被应用到数据分析挖掘领域中，由于理论简单易懂，相对于深度学习而言，应用中不需要更多的训练方法和技巧，所以铺开得非常快，在数据分析的各种行业中都有应用。

在深度学习中，有不止一层隐层，所以在特征的选取、隐层的确定、单层训练、回归训练等方面都遇到了理论和应用的难题，坦白说，一直到 2006 年之前，都没有广阔的应用。

在深度学习之前，有这些浅层学习方法表征不了或解决不了的事情吗？我们还是以图像识别举例说明，如图 3-15 所示。

这个图分为三层，最下面一层是我们的基础方块，如果是浅层学习，在只有一层的情况下会出现什么结果呢？如果我们将中间这层去除，则最底下的基本线条看起来完全一致；如果要从基础线条中识别出脸、汽车、大象、椅子等，则需要不断地手工更改参数，也许一次训练还不够，需要很多次训练，才有可能训练

出只区分出脸和汽车的模型。

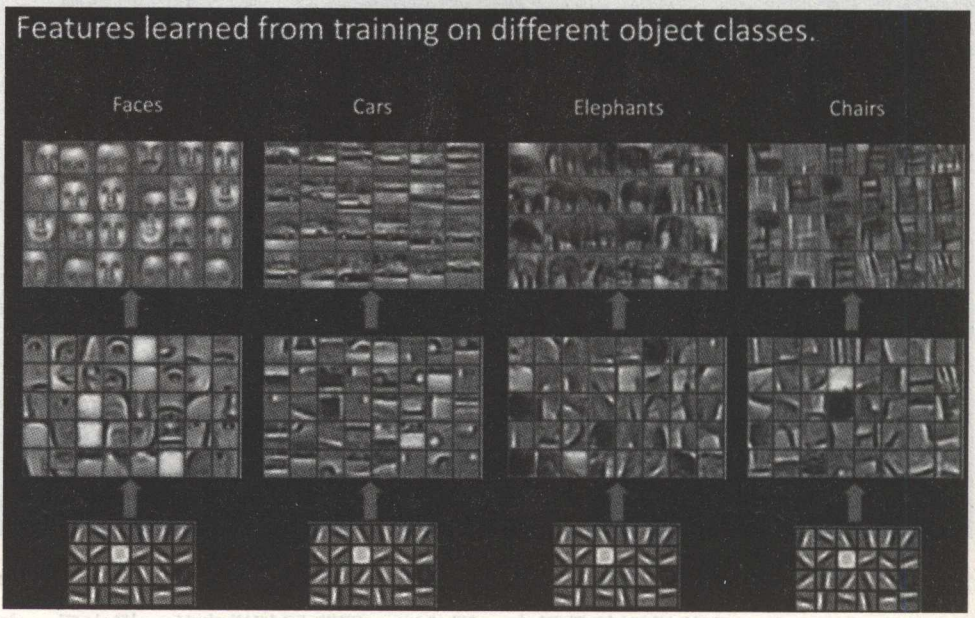


图 3-15　不同物体的图像识别

再比如，我们将底层的基本线条去掉，只保留上面两层。注意，这时需要人工调整参数的工作不多，但这样的数据准备怎么办？靠什么方式在一副完整的人物照片或风景照片中切分出中间那层的方块图像？别告诉我用人工的方式切分。即使你能切分，你需要准备多少训练数据？脸和汽车的数据在中间层就有明显的区别了。是不是每种样子的汽车轮胎或者每种人类的眼睛你都需要准备好数据呢？

这种工作中需要调整的参数过多，需要的样本数量巨大，且一定需要人工干预，这就是深度不够的缺陷。

刚才我们以 2006 年为界，说在 2006 年之前，深度学习没有什么应用，在 2006 年之后深度学习的应用就多了起来，那 2006 年发生了什么呢？

2006 年，以 Hinton 为首的研究人员在深度信念网络（Deep Belief Network，

DBN）方面的划时代性的工作，将此问题终结。其代表性的论文是：

• Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets. Neural Computation. 18:1527-1554, 2006.

• Yoshua Bengio, Pascal Lamblin, Dan Popovici and Hugo Larochelle, Greedy Layer-Wise Training of Deep Networks, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems 19 (NIPS 2006), pp. 153-160, MIT Press, 2007.

• Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra and Yann LeCun. Efficient Learning of Sparse Representations with an Energy-Based Model, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems (NIPS 2006), MIT Press, 2007.

注意这些论文中提出了以下几个非常关键的原则。

（1）非监督学习被用来（预）训练各个层。

（2）非监督学习在之前学习到的层次之上，一次只学习一个层次，每个层次学习到的结果将作为下一个层次的输入。

（3）除了一些专门用于预测的层次外，用监督学习来调整层与层之间的权重。

3.4 深度学习和神经网络

如图 3-16 所示，深度学习网络与传统的神经网络之间的相同之处，就是二者都是相似的分层结构，包括输入层、隐层和输出层的多层网络，其中只有相邻层之间有链接，同一层及跨层之间是没有连接的。不同之处在于，传统神经网络一般只有两层至三层的神经网络，参数和计算单元有限，对复杂函数的表示能力有限，学习能力也有限；而深度学习具有五层至十层，甚至更多的神经网络，并且引入了更有效的算法。深度网络这种分层结构，比较接近人类大脑的结构（但不得不说，实际上相差得还是很远，考虑到人脑是个异常复杂的结构，很多机理我

们目前都未知）。

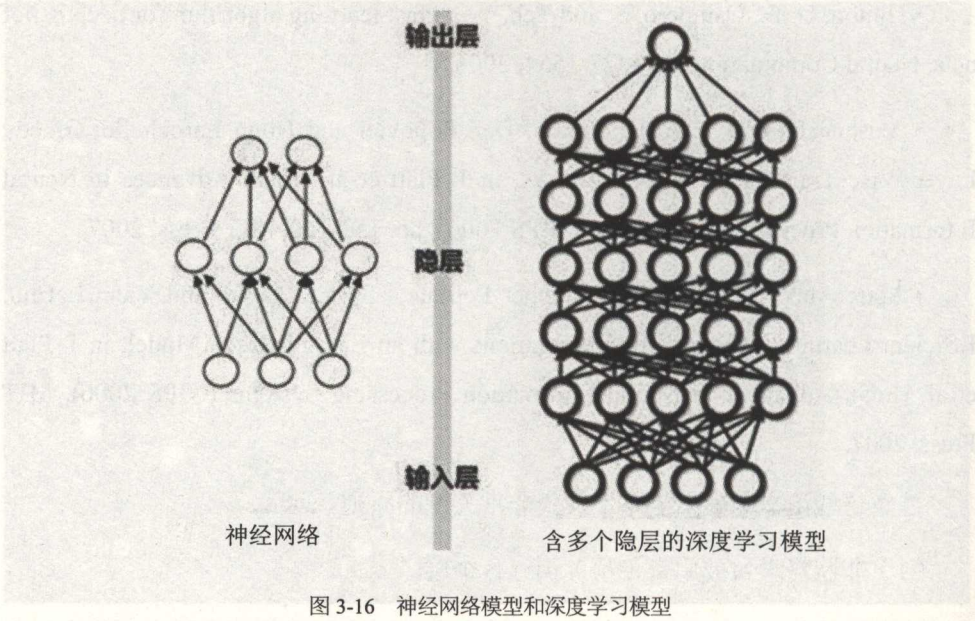


图 3-16　神经网络模型和深度学习模型

3.5　如何训练神经网络

到目前为止，我们已经对深度学习的概念和优点有了一定了解，如何训练一个深度模型才是我们要重点解决的问题。

我们在讲深度模型的训练方法时，先重温神经网络那章讲的内容：我们已经讨论了神经元是由什么构成的，神经元的输入和输出如何计算，后面更谈到了如何计算误差，权值的重新计算，如何纠正误差等。这是感知机的训练方法。

对于包含多个隐层的感知机（多层神经网络）我们如何训练呢？

3.5.1　BP 算法：神经网络训练

我们在前面章节中解决了神经网络的 XOR 逻辑运算问题，当时遗留了一个

概念没有解释——BP 算法。什么是 BP 算法呢？BP 的全称是 Back Propagation，也称为 Error Back Propagation，意思是误差反向传播，也算说明了 BP 算法的特点。

在多层感知机中，我们有一个普遍的问题，就是多层感知机如何获取隐层的权值的问题。在单个神经元计算中，讲到了用实际输出结果和期望输出结果的误差来调整单个神经元的权值。能否借用一下这个想法，通过输出层得到输出结果和期望输出的误差，间接调整隐层的权值呢？

BP 算法的基本想法是：由信号正向传播和误差反向传播。

（1）信号正向传播：正向传播时，输入样本从输入层传入，经各隐层逐层处理后，传向输出层。若输出层的实际输出与期望的输出（教师信号）不符，则转入误差的反向传播阶段。

（2）误差反向传播：将输出以某种形式通过隐层向输入层逐层反传，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，此误差信号即作为修正各单元权值的依据。

这里不过多地展开 BP 训练方法，读者要明白 BP 算法的核心思想是在反向传播上将所得误差分摊给各层所有的单元。注意，这里是所有单元！

3.5.2 BP 算法的问题

刚才我们说了 BP 算法，大家认为我们在深度学习中也用它吗？不完全是，在这里提起 BP 算法是要在深度学习中讨论它的缺点，引出适合深度学习的训练过程。我们先来看看 BP 算法作为传统多层感知机训练方法的问题。

5 层以内的神经网络，我们可以用 BP 算法训练，5 层以上的神经网络用 BP 算法训练就很不理想了。深度结构（涉及多个非线性处理单元层）非凸目标函数中普遍存在的局部最小是训练困难的主要来源。

BP 算法存在如下问题。

（1）梯度越来越稀疏：从顶层往下，误差校正信号越来越小。

（2）收敛到局部最小值：尤其是从远离最优区域开始的时候（随机值初始化会导致这种情况的发生）。

（3）一般，我们只能用有标签的数据来训练：大部分数据是没标签的，而大脑可以从没有标签的数据中学习。

通过梯度下降方法在训练过程中修正权重使得网络误差最小。在层次深的情况下，性能变得很不理想[传播时容易出现所谓的梯度扩散（Gradient Diffusion）]或称之为梯度消失，根源在于非凸目标函数导致求解陷入局部最优，且这种情况随着网络层数的增加更加严重，即随着梯度的逐层消散，导致其对网络权重调整的作用越来越小，所以只能转而处理浅层结构（小于等于 3），这就限制了性能。

除了 BP 算法，我们有其他数据模型和算法吗？科学家当然没有闲着，如上文所述，很多浅层模型相继提出，其中就包括鼎鼎有名的 SVM 支持向量机。我们将在第 4 章详细讨论深度学习的各种算法。

3.6 总结深度学习及训练过程

现实生活中，人们为了解决一个问题，如对事物的分类（事物可以是文档或图像等），首先必须做的是搞清楚怎么表达这个事物，比如这个事物如果是动物的话，它有没有羽毛？有几条腿？卵生还是胎生？这些都将是我们选取的特征，大家可以参考本书2.6节里谈到的识别动物时我们是如何选取特征的内容。然后，我们才能做动物的分类或者水果的分类。在图像处理中，我们可以用像素的集合体，也就是一块图像来表示完整的图像。这时，特征选取的好坏对于分类或者预测的结果影响是非常大的。因此，选取一个什么特征，怎么选取一个特征对于解决实际问题非常重要。

然而，人为地选取特征是一件耗时耗力且面对大量未知的东西没有什么规律可循的方法，能不能选取好很大程度上靠经验和运气。比如，我们做水果分类时，选取香蕉和苹果的特征很容易，这是因为人类已经具有了相当的生活经验。如果

让一个完全没见过苹果和香蕉的人选取特征，他会怎么选取呢？甚至让你选取一些特征去区别一个红富士苹果或山东国光苹果你都无从下手。既然手工选取特征不太好进行，能不能让机器自动地学习一些特征呢？答案是能！深度学习就是用来干这个事情的（它还有个别名 Unsupervised Feature Learning），Unsupervised 的意思就是非监督式。因此，自动地学习特征的方法，统称为深度学习。

我们重新定义下深度学习。准确地说，深度学习首先利用无监督学习对每一层进行逐层预训练（Layerwise Pre-Training）去学习特征；每次单独训练一层，并将训练结果作为更高一层的输入；然后到最上层改用监督学习从上到下进行微调（Fine-Tune）去学习模型。

深度学习训练的具体过程如下。

（1）使用自下而上的非监督学习（就是从底层开始，一层一层地往顶层训练）。

采用无标定数据（有标定数据也可）分层训练各层参数，这一步可以看作是一个无监督训练过程，是和传统神经网络区别最大的地方，这个过程可以看作是特征学习的过程。

具体来讲，先用无标定数据训练隐层的最底层，训练时先学习最底层的参数（这一层可以看作是得到一个使得输出和输入差别最小的三层神经网络的隐层），由于模型能力的限制及稀疏性约束，使得得到的模型能够学习到数据本身的结构，从而得到比输入更具有表示能力的特征；在学习得到第 n-1 层后，将 n-1 层的输出作为第 n 层的输入，训练第 n 层，由此分别得到各层的参数。

（2）自顶向下的监督学习，就是通过带标签的数据去训练，误差自顶向下传输，对网络进行微调。

基于第一步得到的各层参数进一步 fine-tune 整个多层模型的参数，这一步是一个有监督训练过程；第一步类似神经网络的随机初始化初值过程，由于深度学习的第一步不是随机初始化，而是通过学习输入数据的结构得到的，因此这个初值更接近全局最优，能够取得更好的效果；所以深度学习效果好很大程度上归功于第一步（特征学习的过程）。

这个过程如图 3-17 所示。

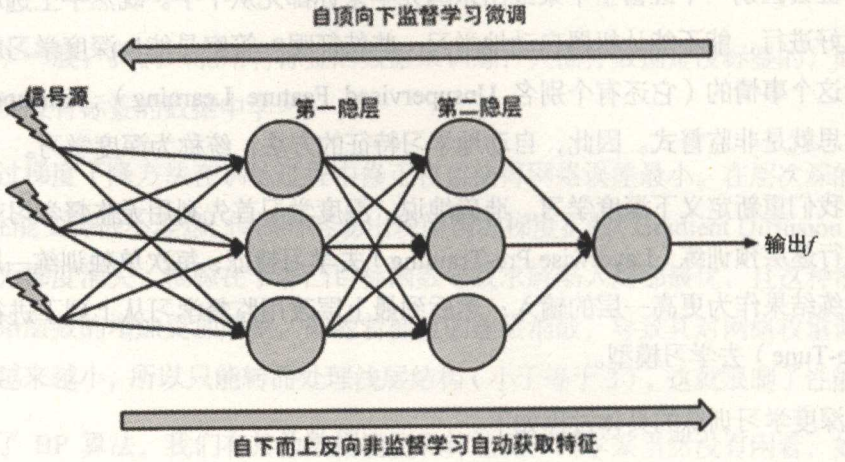


图 3-17　特征学习过程

第 3 章中我们了解了深度学习的概念，明白了深度学习是属于机器学习的一类，学习了特征及特征粒度的提取。这里为什么要讲 BP 算法，BP 算法的思想（反向传播，修正权值）其实和我们说的反向非监督学习自动获取特征的思想是相同的。第 4 章中，我们会详细谈谈深度学习经常会用到的一些方法。

4 深度学习的常用方法

2008 年 6 月，《连线》杂志主编克里斯·安德森（Chris Anderson）发表文章，题目是《理论的终极，数据的泛滥将让科学方法过时》。文中引述经典著作《人工智能的现代方法》的合著者，当时还是谷歌研究总监的彼德·诺米格（Peter Norvig）的话，“一切模型都是错的。进而言之，抛弃它们，你就会成功”。

言下之意，精巧的算法是无意义的。面对海量数据，即便只用简单的算法，也能得到出色的结果。与其钻研算法，不如研究云计算，处理大数据。

如果这番言论发生在 2006 年以前，谁都无法反驳。但是自 2006 年以来，机器学习领域取得了突破性的进展，请参考我们前面提到的自 2006 年以来深度学习学术界的大突破。

现在我们可以说，要得到出色的结果不仅依赖于云计算及 Hadoop 框架的大数据的并行处理能力，而且依赖于算法。这个算法就是深度学习。

请大家注意，深度学习这个概念和其他机器学习算法最大的不同在于如何找到特征，而特征的抽取过程就是一个抽象的过程。以往，特征抽取的其他机器学习算法都是对一类问题有解，比如 K-Means 等聚类。深度学习抽象模拟了人类神

经元传递信息和链接的方式，理论上讲，它可以完全搞定人类多种分类和预测问题，甚至可以研究未知领域，可以成为一个专家，一个科学家。深度学习是不是那个“奇点”呢？不知道，但反正人类更靠近它了。

4.1 模拟大脑的学习和重构

小明是一名小学生，他正在学习简单的英语，今天学习到了 Easy 这个词。通过老师的讲解和书本上的知识，小明知道了 Easy 对应的汉语是“容易”的意思，他将整个词的形状和翻译都记录下来。第二天，老师开始检查小明的学习成果，小明在黑板上写下“容易”→“easy”。老师很高兴，给了他一个对钩。

请大家注意，这里小明写出的 easy 和昨天学到的“Easy”有偏差，或者叫误差，就是首字母变成小写了！小明将学习到的“Easy”记下来，再次反馈出来时，没有严格地区分首字母大小写，写出了“easy”这个单词。严格意义上讲，昨天和今天的“Easy”单词是有差异的（首字母大小写）。

小明昨天学习的过程就是编码过程，今天将整个单词还原出来就是解码过程，虽然解码的结果“easy”和昨天学习的“Easy”有误差，少了首字母大写，但这种误差是可以接受的，所以这就是自动编码器（AutoEncoder）的思想！

从生物学大脑角度考虑，学习和重构就像编码和解码一样。

我们把小明学习的过程抽象下：假设我们有一个系统 S，它有 n 层（$S_1,S_2,\cdots,S_n$），它的输入是 I，输出是 O，形象地表示为：$I>S_1>S_2>\cdots>S_n>O$，如果输出 O 等于输入 I，即输入 I 经过这个系统变化之后没有任何的信息损失，保持了不变，这意味着输入 I 经过每一层 S_i 都没有任何的信息损失，即在任何一层 S_i，它都是原有信息（即输入 I）的另外一种表示。假设我们有一堆输入 I（如一堆图像或者文本），我们设计了一个系统 S（有 n 层），我们通过调整系统中的参数，使它的输出仍然是输入 I，那么我们就可以自动地获取得到输入 I 的一系列层次特征，

即 S_1，S_2，…，S_n。

另外，前面是假设输出严格地等于输入，这个限制太严格，我们可以略微放松这个限制，例如我们只要使得输入与输出的差别尽可能小即可。

继续往下看。

4.1.1 灰度图像

假设这里有一个由 28 像素 × 28 像素的灰度图像组成的训练集，且每一个像素的值都作为一个输入层神经元的输入（这时输入层就会有 784 个神经元）。输出层神经元要有相同的数目（784），且每一个输出神经元的输出值和输入图像的对应像素灰度值相同。

在这样的算法架构背后，神经网络学习到的实际上并不是一个训练数据到标记的“映射”，而是去学习数据本身的内在结构和特征［隐含层也被称作特征探测器（Feature Detector）］。通常隐含层中的神经元数目要比输入/输出层少，这是为了使神经网络只学习最重要的特征并实现特征的降维。

如图 4-1 所示，第一隐层提取线段特征，第二隐层提取脸部局部特征，第三隐层提取全脸特征。

我们想在中间层用很少的节点在概念层上学习数据，产生一个适合人脸识别的表示方法。每一个隐层就是特征的提取（反向提取特征），最上层，也就是输入层也是一种维度的特征，这种维度是将图片的每个点、每个颜色都放进去作为特征传递给下一层的。

从输入一层层下去，我们每个隐层比上一层减少若干神经元，然后再反向训练权值。这样我们最后学习的东西也就是几十种脸的类型了，极大地减少了特征量，这就是自编码器做的事情。

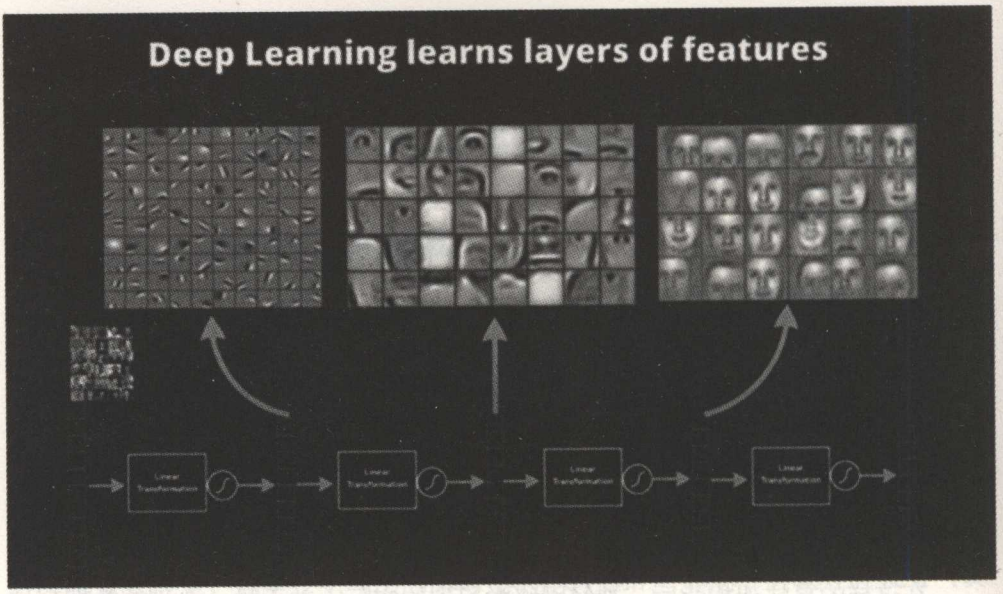


图 4-1　特征学习每层的表征

4.1.2　流行感冒

为了更好地描述自编码器，我们再略微深入进去，先看一个应用。

我们使用一个简单的数据集，表示感冒的症状。

数据结构如下：

- 输入数据一共六个二进制位。
- 前三位是病的征状。例如，100000 代表病人发烧；010000 代表病人咳嗽；110000 代表病人既咳嗽又发烧等。
- 后三位表示免疫能力，如果一个病人有这个，代表他/她不太可能患此病。例如，000100代表病人接种过流感疫苗。一个组合是010100，代表一个接种过流感疫苗的咳嗽病人，等等。

我们定义了一个规则：当一个病人同时拥有前三位中的两位时，我们认为他生病了；如果至少拥有后三位中的两位，那么他是健康的，如：

- 111000, 101000, 110000, 011000, 011100 =生病
- 000111, 001110, 000101, 000011, 000110 =健康

接着来训练一个自编码器（使用反向传播），六个输入、六个输出神经元，而只有两个隐含神经元。

在经过几百次迭代以后，我们发现每当一个“生病”的样本输入时，两个隐含层神经元中的一个总是显示出更高的激活度。输入一个“健康”样本时，另一个隐含层神经元会显示更高的激活度。

本质上来说，这两个隐含神经元从数据集中学习到了流感症状特征的一种压缩表示方法。为了检验它是不是真的实现了学习，我们再看过度拟合的问题。通过训练我们的神经网络学习到的是一个紧致且简单的表示方法，而不是一个高度复杂且对数据集过度拟合的表示方法。

某种程度上，与其说在找一种简单的表示方法，自动编码器更是在尝试从“感觉”上去学习数据。

4.1.3 看看如何编解码

让我们一步步深入地看如何编解码。

设计一个自编码器，假设 input 有 4 个不同输入，则表示为：

(0,0,0,1)记作 1

(0,0,1,0)记作 2

(0,1,0,0)记作 3

(1,0,0,0)记作 4

因为所有的 input 只有这 4 种，所以其实用 4 个 bit 是不经济的或者叫不紧致的，存在压缩表示的可能性，比如 2 个 bit 就可以表示这 4 个不同的数。

那么，我们设计了输入层是个单元（4 个单元接受 4bit 编码的 input），隐藏

层 2（因为我们已经知道 2 个 bit 就够了，所以 2 个隐藏层，具有足够的表达能力）；输出层 4（为了能让输出和输入保持一致）。

我们用 Keras 定义编解码器，因为最终的输出都要大于 0，所以我们使用 Sigmoid 激活函数。

```
encoder = containers.Sequential([Dense(2, input_dim=4,
activation='sigmoid')])
decoder = containers.Sequential([Dense(4, input_dim=2,
activation='sigmoid')])
```

通过 60000 次迭代，我们看到如下情况：

(1,0,0,0)→(0.948,0.025)→(0.892,0.096,0.116,0.0008)

(0,1,0,0)→(0.044,0.032)→(0.102, 0.848,0.004,0.108)

(0,0,1,0)→(0.958,0.966)→(0.066,0.001,0.858,0.056)

(0,0,0,1)→(0.019,0.930)→(0.001,0.001,0.136,0.884)

input_layer　　hidden_layer　　output_layer

我们可以看到，（0,0,0,1）通过自编码的压缩表示变成了（0.019,0.930），但（0.019,0.930）能否表示（0,0,0,1）呢？我们将它解码，得到了（0.001,0.001,0.136,0.884），可以表示原有输入（0,0,0,1）的近似值。这样，我们认为这个自编码器是成功的，并且可以将一个 4 特征值降为一个 2 特征值来表达，可以减少一半的特征。

hidden 层的编码恰好可以看作是：

(0.948,0.025) 1,0

(0.044,0.032) 0,0

(0.958,0.966) 1,1

(0.019,0.930) 0,1

输入的（0,0,0,1）可以被（1,0）紧凑表示，最终 4bit 的信息，可以用 2bit 表示，在输入足够复杂时，压缩表示是有价值的。我们对（0,0,0,1）提取的特征就

是（0.019,0.930）。

总结：首先，利用压缩方法使输入的数据降维，变成隐层的东西。然后，我们将它还原出来，得到一个近似于输入的输出值，比如第一个输入（1,0,0,0）输出是（0.892,0.096,0.116,0.0008），在整个过程中这个自编码器自然学习了特征，对于输出而言，我们发现和输入是有差别的，这个差别可以被用作标定数据去训练隐层。

4.1.4 如何训练

以上说的都是如何选取特征，本节讲怎么利用输入输出的差异去训练，以及为什么要这么做。

在有监督的学习方法里，我们给定了一组输入数据，给定了数据的答案，或者叫标签。我们就能根据最后答案的不同来训练这个算法，例如，训练一个分类器，如果样本数量足够多，覆盖足够全，我们能训练出某种可以完成我们要求的分类器，如图 4-2 所示。

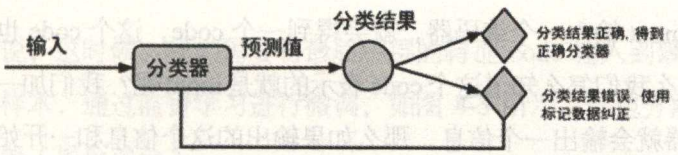


图 4-2 有监督分类器

如果我们没有样本结果，没有标签了，该怎么办？

如图 4-3 所示，我们无法得到误差值，所以无法继续优化分类器参数。自动编码器就是要解决无答案数据时，如何得到误差，进而纠正其中的值的问题。

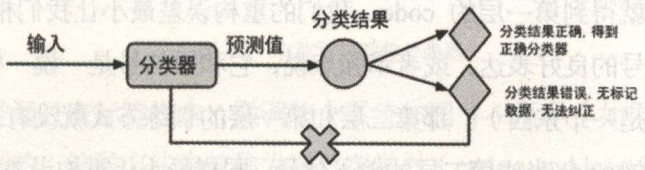


图 4-3 有监督分类器，无标记数据无法纠正

再来看看之前小明学习英文单词“Easy”的例子，这里我们假设小明学习错了，将“Easy”编码学成了“easy”，在解码时将“easy”转换成“eacy”，第二天老师考察小明，小明将“容易”解释成“eacy”，老师判断这是错的，因为输出值无法和输入值匹配，经过学习表达出来的“eacy”是错误的！这里老师其实做的就是拿原始输入的“Easy”单词和“eacy”做比较得到错误的结论，并没有利用任何外来的数据或知识结构。接着，小明开始重新学习的过程，第三天小明将“Easy”学习成了“easy”并把“easy”关联到“轻松容易”的场景上。这样最终表达出来的值我们认为和“Easy”的原始输入是一致的，如图 4-4 所示。

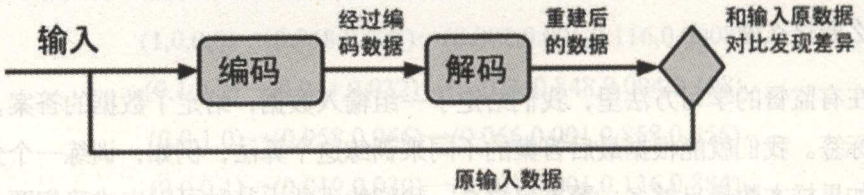


图 4-4　编解码过程

我们将 input 输入一个编码器，就会得到一个 code，这个 code 也就是输入的一个表示，那么我们怎么知道这个 code 表示的就是 input 呢？我们加一个解码器，这时候解码器就会输出一个信息，那么如果输出的这个信息和一开始的输入信号 input 是很像的（理想情况下就是一样的），那很明显，我们就有理由相信这个 code 是靠谱的。所以，我们就通过调整 encoder 和 decoder 的参数，使重构误差最小，这时我们就得到了输入 input 信号的第一个表示，也就是编码 code。因为是无标签数据，所以误差的来源就是直接重构后与原输入相比得到。

上面我们就得到第一层的 code，我们的重构误差最小让我们相信这个 code 就是原输入信号的良好表达，或者牵强点说，它和原信号是一模一样的（表达不一样，反映的是一个东西）。那第二层和第一层的训练方式就没有差别了，我们将第一层输出的 code 当成第二层的输入信号，同样最小化重构误差，就会得到第

二层的参数，并且得到第二层输入的 code，也就是原输入信息的第二个表达。其他层用同样的方法炮制就行（训练这一层，前面层的参数都是固定的，并且它们的 decoder 已经没用了，都不需要了）。

4.1.5 有监督微调

用上面的方法，我们就可以得到很多层。至于需要多少层（或者深度需要多少，这本身就没有一个科学的评价方法）需要自己试验调。每一层都会得到原始输入的不同的表达。当然，我们觉得它是越抽象越好，就像人的视觉系统一样。

到这里，这个自动编码器还不能用来分类数据，因为它还没有学习如何去连接一个输入和一个类。它只是学会了如何重构或者重复它的输入而已。或者说，它只是可以学习一个代表输入的特征，这个特征可以最大程度上代表原输入信号。那么，为了实现分类，我们就可以在自动编码器的最顶的编码层添加一个分类器（例如，罗杰斯特回归、SVM 等），然后通过标准的多层神经网络的监督训练方法（梯度下降法）去训练。

也就是说，这时候，我们需要将最后一层的特征 code 输入到最后的分类器，通过有标签样本，通过监督学习进行微调，如图 4-5 所示，这也分两种，一种是只调整分类器（黑色部分）。

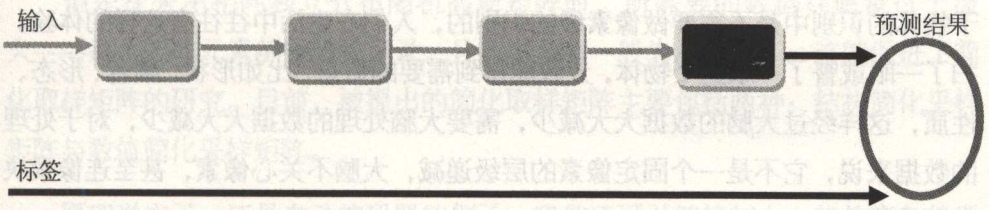


图 4-5 有监督只调整分类器

另一种是通过有标签样本，微调整个系统如图 4-6 所示［如果有足够多的数据，则是最好的。这种方法也被称为端对端学习（end-to-end learning）］。

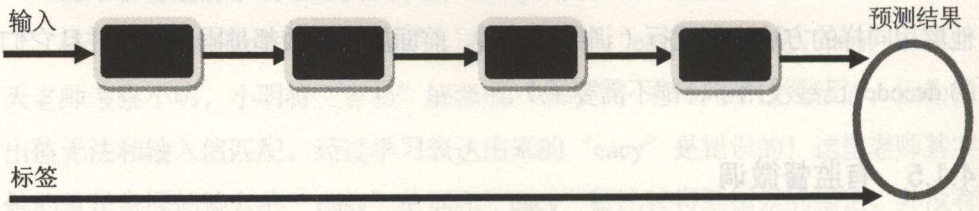


图 4-6　有监督微调整个系统

一旦监督训练完成，这个网络就可以用来分类了。神经网络的顶层可以作为一个线性分类器，然后我们可以用一个更好性能的分类器取代它。

在研究中发现，如果在原有的特征中加入这些自动学习得到的特征可以大大提高精确度，甚至在分类问题中比目前最好的分类算法效果还要好！

4.2　快速感知：稀疏编码（Sparse Coding）

前文提到，人工神经网络实际上是模拟生物神经网络，我们再回过头来看看生物神经网络的特性——主要是视觉特性。

人在获取图像信息的时候，我们不是像计算机那样一个像素一个像素地读，这样的读取方式往往仅限于机器处理图像的方式，不管是读取还是存储都仅限于像素级，这种读取方式会把所有的信息都过滤一遍，但生物在识别或者说人类对于物体的识别中是不需要做像素级的识别的，人们在生活中往往有这样的体验。扫了一眼或瞥了一眼这个物体，大致能得到需要的信息，比如形状、颜色、形态、性质，这样经过大脑的数据大大减少，需要大脑处理的数据大大减少，对于处理的数据来说，它不是一个固定像素的层级递减，大脑不关心像素，甚至连像素块都不关注。

从用眼睛扫一眼物体到大脑感知物体并做出判断，这个时间非常短。初期，做视觉识别的科学家很感兴趣，大脑为什么能这么快地处理图像或者说动态视频呢？大脑是按照什么规则过滤像素的呢？很自然，科学家们都想将这个原理搞清

楚，并弄到机器上去，让机器也像人脑那么快地识别物体并做出相应的反馈。

在看稀疏编码之前，我们先来看一个新名词。

压缩感知

信号处理领域中的一个常见问题就是从一系列的采样中重建原本的信号。大致而言，由于不可能重建出未被采样的部分信号，所以这一任务是无法完全实现的。然而，通过借助对于信号（性质）的预先了解或合理假设，完美地通过一系列采样重建原信号就成为了可能。随着科学的发展，数学家们逐步增进了如何做出合理假设的能力，并慢慢了解到在何种情况下可将这些假设一般化、推广化。

从上面这段维基百科的解释中可以看到，信号处理领域的第一个问题就是采样，第二个问题就是重建完整信号。我们先来看看第一个问题。

如何采样

理论上，为了确保信号重建的准确度，需要令所采用的取样矩阵各行列之间相干性尽量低，且须矩阵元素取值随机性尽量高。

考虑到以上原因，最为标准的做法是采用相同独立分布随机高斯矩阵（identical independent distributed random Gaussian matrix）对待处理信号进行取样，即可确保在信号具有足够稀疏性的前提下得到较佳的压缩感知重建效果。

但是在采用相同独立分布随机高斯矩阵时，所需要的数据存储量过于庞大——每个矩阵元素都要单独记录，且数据类型一般为浮点数——这就促进了简化取样矩阵的研究。目前，被提出的简化取样矩阵主要包括两种：结构简化采样矩阵与数值简化采样矩阵。

原理说完了，还是自动编码器的例子，我们还可以继续加上一些约束条件得到新的深度学习方法：如果在自动编码器的基础上加上 L1 的规则限制（L1 主要是约束每一层中的节点中大部分都要为 0，只有少数不为 0，这就是 Sparse 名字的来源），我们就可以得到 SparseAutoEncoder 法。

如果隐藏节点比可视节点（输入、输出）少的话，被迫的降维，自编码器会自动习得训练样本的特征（变化最大，信息量最多的维度）。但是如果隐藏节点数目过多，甚至比可视节点数目还多的时候，自编码器不仅会丧失这种能力，更可能会习得一种“恒等函数”——直接把输入复制过去作为输出。这时，我们需要对隐藏节点进行稀疏性限制。

所谓稀疏性，就是对一对输入图像，隐藏节点中被激活的节点数（输出接近1）远远小于被抑制的节点数目（输出接近0）。那么，使神经元大部分的时间都是被抑制的限制被称作稀疏性限制。

如图 4-7 所示，其实就是限制每次得到的表达编码尽量稀疏。因为稀疏的表达往往比其他的表达要有效（人脑好像也是这样的，某个输入只是刺激某些神经元，其他大部分的神经元是受到抑制的）。

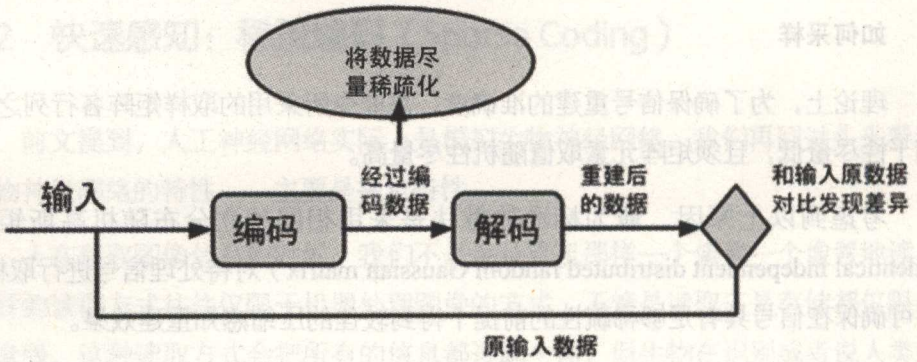


图 4-7　编解码中的稀疏表达

4.3　栈式自编码器

栈式自编码器是一个由多层稀疏自编码器组成的神经网络，如图 4-8 所示，其前一层自编码器的输出作为其后一层自编码器的输入。对于一个 n 层栈式自编码器的编码过程就是，按照从前向后的顺序执行每一层自编码器的编码步骤。

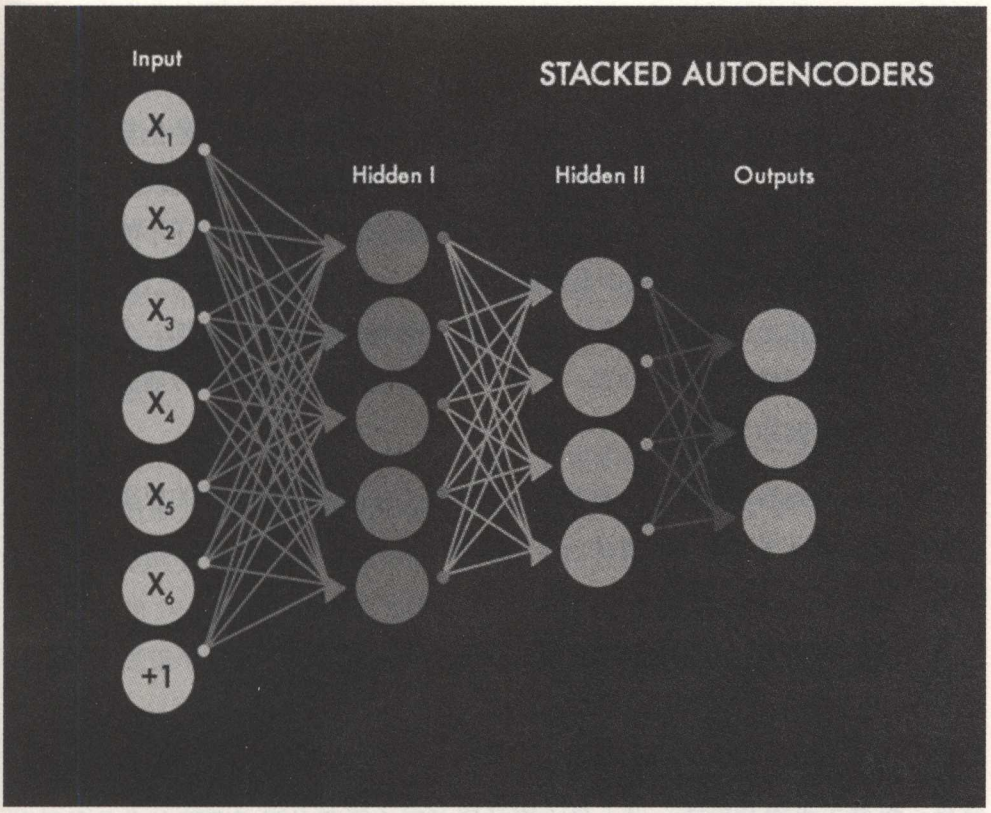


图 4-8 栈式自编码器结构

自编码器的隐含层 t 会作为 t+1 层的输入层。第一个输入层就是整个网络的输入层。利用贪心法训练每一层的步骤如下。

（1）通过反向传播的方法，利用所有数据对第一层的自编码器进行训练（t=1，图中的左数第二层连接部分）。

（2）训练第二层的自编码器 t=2（左数第三层连接部分）。由于 t=2 的输入层是 t=1 的隐含层，所以我们已经不再关心 t=1 的输入层，可以从整个网络中移除。整个训练开始于将输入样本数据赋到 t=1 的输入层，通过前向传播至 t=2 的输出层。下面 t=2 的权重（输入→隐含和隐含→输出）使用反向传播的方法进行更新。t=2 的层和 t=1 的层一样，都要通过所有样本的训练。

（3）对所有层重复步骤一和步骤二（即移除前面自编码器的输出层，用另一个自编码器替代，再用反向传播进行训练）。

（4）步骤一至步骤三被称为预训练，这将网络里的权重值初始化至一个合适的位置。但是通过这个训练并没有得到一个输入数据到输出标记的映射。例如，一个网络的目标是被训练用来识别手写数字，经过这样的训练后还不能将最后的特征探测器的输出（即隐含层中最后的自编码器）对应到图片的标记上去。这样，一个通常的办法是在网络的最后一层（即蓝色连接部分）后面再加一个或多个全连接层。整个网络可以被看作是一个多层的感知机，并使用反向传播的方法进行训练（这步也被称为微调）。

栈式自编码器，提供了一种有效的预训练方法来初始化网络的权重，这样你得到了一个可以用来训练的复杂、多层的感知机。

4.4 解决概率分布问题：限制波尔兹曼机

4.4.1 生成模型和概率模型

1. 决策函数和条件概率分布

我们在第 3 章讲机器学习的时候，提到一种分类方式是按照是否有“老师”分类，这样的分类方法就好像有人跟你说了一个问题，并同时告诉你答案是一样的。你先按照自己的方法得出答案，如果和别人给出的答案不一样，再调整，直到差不多（收敛）为止，而这种学习方式就是有监督学习。有监督学习用来从已知数据中学习一个模型。我们测试该模型，一般输入 X 而输出为预测值或分类值 Y。这个模型的一般形式为决策函数 $Y=f(X)$或者条件概率分布 $P(Y|X)$。

什么叫决策函数呢？你输入一个 X，决策函数输出一个 Y，拿这个 Y 与一个阈值比较，根据比较结果判定 X 属于哪个类别。例如，两类（w_1 和 w_2）分类问题，如果 Y 大于阈值，那么 X 就属于类 w_1；如果小于阈值就属于类 w_2。这样就得到

了该 X 对应的类别。

什么叫条件概率分布？你输入一个 X，它通过比较其所属类的概率，输出概率最大的那个作为该 X 对应的类别。例如，$P(w_1|X)$输出一个值，表示如果为条件 X 的话，成为 w_1 这个类型的概率有多大？我们将 $P(w_1|X)$和 $P(w_2|X)$的值比较，如果 $P(w_1|X)$大于 $P(w_2|X)$，那么我们就认为 X 是属于 w_1 类的。

我们拿之前的苹果香蕉举例，例如，条件 X 为颜色，输入如果是黄色的话，为香蕉的概率就最大，这里就可以用 $P(w_1|X)$的值大于 0.5 表示为香蕉的概率比为苹果的概率大。

上面两个模型都可以实现对给定的输入 X 预测相应的输出 Y 的功能。实际上，通过条件概率分布 $P(Y|X)$进行预测也是隐含着表达成决策函数 $Y=f(X)$的形式的。例如，也是两类 w_1 和 w_2，我们求得了 $P(w_1|X)$和 $P(w_2|X)$，那么实际上判别函数就可以表示为 $Y=P(w_1|X)/P(w_2|X)$，这样就能将 $P(w_1|X)/P(w_2|X)$作为一个阈值，如果 Y 大于 1 或者这个阈值，那么 X 就属于类 w_1，如果小于阈值就属于类 w_2。

我们用决策函数表示概率分布模型，很神奇的事是，竟然反过来也可以，也就是说 $P(Y|X)$概率分布也可以表示决策函数。因为一般决策函数 $Y=f(X)$是通过学习算法使你的预测和训练数据之间的误差平方最小化，而贝叶斯告诉我们，虽然它没有显式地运用贝叶斯或者以某种形式计算概率，但它实际上也是在隐含地输出极大似然假设（MAP 假设）。也就是说，学习器的任务是在所有假设模型有相等的先验概率条件下，输出极大似然假设。

这里我们总结下，**概率分布和决策函数是可以互相转化的**。

所以，分类器的设计有时候就是在给定训练数据的基础上估计其概率模型 $P(Y|X)$。如果可以估计出来，那么就可以分类。听明白了吗？**决策函数在分类器的设计时就需要估计概率模型**。一般来说，概率模型是比较难估计的。给你一堆数，特别是数不多的时候，靠人工区分很难找到这些数满足什么规律。

那能否不依赖概率模型直接设计分类器呢？事实上，分类器就是一个决策函数（或决策面），如果能够从要解决的问题和训练样本出发直接求出判别函数，

就不用估计概率模型了，这就是决策函数 $Y=f(X)$的伟大使命。例如支持向量机，我已经知道它的决策函数（分类面）是线性的了，也就是可以表示成 $Y=f(X)=WX+b$ 的形式，那么我们通过训练样本来学习得到 W 和 b 的值就可以得到 $Y=f(X)$了。还有一种更直接的分类方法，它不用事先设计分类器，而是确定分类原则，根据已知样本（训练样本）直接对未知样本进行分类。包括近邻法，它不会在进行具体的预测之前求出概率模型 $P(Y|X)$或者决策函数 $Y=f(X)$，而是**在真正预测的时候，将 X 与训练数据的各类的 X_i 比较，和哪些比较相似，就判断这个 X 也属于 X_i 对应的类**。

2. 生成方法和判别方法

监督学习方法又分生成方法（Generative Approach）和判别方法（Discriminative Approach），所学到的模型分别称为生成模型（Generative Model）和判别模型（Discriminative Model）。咱们先谈判别方法，因为它和前面说的都差不多，比较容易明白。

判别方法：由数据直接学习决策函数 $Y=f(X)$或者条件概率分布 $P(Y|X)$作为预测的模型，即判别模型。基本思想是有限样本条件下建立判别函数，不考虑样本的产生模型，直接研究预测模型。典型的判别模型包括 k 近邻、感知机、决策树、支持向量机等。

生成方法：从数据学习联合概率密度分布 $P(X,Y)$，然后求出条件概率分布 $P(Y|X)$作为预测的模型，即生成模型 $P(Y|X)= P(X,Y)/ P(X)$。

基本思想是先建立样本的联合概率密度模型 $P(X,Y)$，然后得到后验概率 $P(Y|X)$，再利用它进行分类，就像上面说的那样。注意，这里是先求出 $P(X,Y)$才得到 $P(Y|X)$的，然后这个过程还得先求出 $P(X)$。$P(X)$就是你的训练数据的概率分布。

刚才说了，需要你的数据样本非常多的时候，你得到的 $P(X)$才能很好地描述你的数据真正的分布。比如你投硬币，试了 100 次，得到正面的次数和你的试验次数的比可能是 3/10，然后你的直觉告诉你，可能不对，然后你又试了 500 次，

这次正面的次数和你的试验次数的比可能变成 4/10，这时候你半信半疑，不相信上帝还有一只手，所以你又试了 200000 次，这时正面的次数和你的试验次数的比（就可以当成是正面的概率了）就变成 5/10。这时你觉得接近了真实的概率情况。

还有一个问题就是，在机器学习领域有个约定俗成的说法是：不要去学那些对这个任务没用的东西。例如，对于一个分类任务：对一个给定的输入 x，将它划分到一个类 y 中。那么，如果我们用生成模型 $P(X,Y)= P(Y|X)\ P(X)$表示，我们就需要对 $P(X)$建模，但这增加了我们的工作量，让我们觉得耗时比较多（除了上面说的那个估计得到 $P(X)$可能不太准确外）。实际上，因为数据的稀疏性，导致我们都是被强迫地使用弱独立性假设去对 $P(X)$建模的，所以就产生了局限性，所以我们更趋向于直观地使用判别模型去分类。

这样的方法之所以称为生成方法，是因为模型表示了给定输入 X 产生输出 Y 的生成关系，用于随机生成的观察值建模，特别是在给定某些隐藏参数的情况下。典型的生成模型有朴素贝叶斯模型、隐马尔科夫模型等。

3. 生成模型和判别模型的优缺点

在监督学习中，两种方法各有优缺点，适合不同条件的学习问题。

生成方法的特点：前文提到，生成方法学习联合概率密度分布 $P(X,Y)$，所以就可以从统计的角度表示数据的分布情况，能够反映同类数据本身的相似度，但它不关心到底划分各类的那个分类边界在哪里。生成方法可以还原出联合概率分布 $P(Y|X)$，而判别方法不能。生成方法的学习收敛速度更快，即当样本容量增加的时候，学到的模型可以更快地收敛于真实模型；当存在隐变量时，仍可以用生成方法学习。此时判别方法就不能用了。

判别方法的特点：判别方法直接学习的是决策函数 $Y=f(X)$或者条件概率分布 $P(Y|X)$，不能反映训练数据本身的特性，但是它寻找不同类别之间的最优分类面，反映的是异类数据之间的差异。直接面对预测，往往学习的准确率更高。由于直接学习 $P(Y|X)$或 $P(X)$，可以对数据进行各种程度上的抽象、定义特征并使用特征，

因此可以简化学习问题。

我们从生成方法和判别方法讲到了生成模型和判别模型，生成和判别的联系在哪呢？我们可以推出：**由生成模型可以得到判别模型，但由判别模型得不到生成模型**。

再形象点可以吗？

例如，我们有一个输入数据 x，然后我们想将它分类为标签 y（迎面走过来一个人，你告诉我这个人是男的还是女的）。

生成模型学习联合概率分布 $p(x,y)$，而判别模型学习条件概率分布 $p(y|x)$。

下面是个简单的例子。

例如，我们有以下(x,y)形式的数据，x 是特征量：(1,0), (1,0), (2,0), (2,1)

那么 $p(x,y)$是：

	y=0	y=1
x=1	1/2	0
x=2	1/4	1/4

而 $p(y|x)$ 是：

	y=0	y=1
x=1	1	0
x=2	1/2	1/2

我们为了将一个样本 x 分类到一个类 y，最自然的做法就是条件概率分布 $p(y|x)$，我们对其直接求 $p(y|x)$方法叫作判别算法，而生成算法就是求 $p(x,y)$，$p(x,y)$

我们可以通过贝叶斯方法转化为 $p(y|x)$，然后再用其分类。

再假如，你的任务是识别一个语音属于哪种语言。例如，对面一个人走过来，和你说了一句话，你需要识别出她说的到底是汉语、英语还是法语等。那么你可以有两种方法达到这个目的。

（1）学习每一种语言，你花了大量精力把汉语、英语和法语等都学会，我指的学会是你知道什么样的语音对应什么样的语言，然后再有人过来对你开骂，你就可以知道他说的是什么语音，你就可以回骂他了。

（2）不去学习每一种语言，你只学习这些语言模型之间的差别，然后再分类。意思是指我学会了汉语和英语等语言的发音是有差别的，我学会这种差别就好了。

那么，第一种方法就是生成方法，第二种方法是判别方法。

生成算法尝试找出到底这个数据是怎么生成的（产生的），然后再对一个信号进行分类。基于你的生成假设，那么那个类别最有可能产生这个信号，这个信号就属于那个类别。判别模型不关心数据是怎么生成的，它只关心信号之间的差别，然后用差别简单地对给定的一个信号进行分类。这段话有助于你理解判别和生成两种模型，请仔细理解。

4.4.2 能量模型

请一定看完 4.4.1 节再理解此节！

我们在第 0 章讲历史的时候提到了杰弗里·辛顿教授，他在 2006 年提出了一种深度信念网络（Deep Belief Network）并给出了该模型的一个高效学习算法。这个算法成为了其后深度学习算法的主要框架，而这个框架引入了我们之前提到的生成模型，它可以直接自动地从训练集里提取所需要的特征，典型的模型为有限制玻尔兹曼机，自动提取的特征解决了人工提取的考虑不周的因素，而且对于神经网络权重做了个非常重要的初始化，接着可以采用 BP 算法进行分类，实验得出了很好的效果。

在谈 RBM 之前，我们先提一下能量的模型（Energy Based Model），能量模型来源于热动力学，分子在高温中运动剧烈，能够克服局部约束（分子键的一些物理约束，比如键值吸引力等），在逐步降到低温时，分子最终会排列出有规律的结构，此时也是低能量状态。如在一个碗内的小球会停留在碗底，即使受到扰动偏离了碗底，在扰动消失后，它会回到碗底。学过物理的人都知道，**稳态是它势能最低的状态**。因此，稳态对应于某一种能量的最低状态。将这种概念引用到 Hopfield 网络中，Hopfield 构造了一种能量函数的定义。在 RBM 中引进能量函数概念可以进一步加深对这一类动力系统性质的认识，**可以把求稳态变成一个求极值与优化的问题**，由此可以看出将求解的一个问题转换成另一个问题，从而找到一个问题最优解的应用，是我们学习能量模型的意义。

受此启发，早期的模拟退火算法就是在高温中试图跳出局部最小。随机场作为物理模型之一，也引入了此方法。

统计力学的结论表明，**任何概率分布**（注意，概率分布是我们上节讲的东西）**都可以转变成基于能量的模型**，而且很多的分布都可以利用能量模型的特有性质和学习过程，有些甚至从能量模型中找到了通用的学习方法。

了解了能量模型是什么以后，我们再谈谈能量模型和波兹曼机的关系：能量模型需要做的事情就是先定义一个合适的能量函数，然后基于这个能量函数得到变量的概率分布，最后基于概率分布求解一个目标函数（如最大似然）。

那么波兹曼机是什么呢？玻尔兹曼机（Boltzmann machine，BM）是源于物理学的能量函数的建模方法，能够描述变量的高层相互作用。波尔兹曼网络是一种随机网络。描述一个随机网络，总结起来主要有两个结构算法。

第一，概率分布函数。由于网络节点的取值状态是随机的，从贝叶斯网的观点来看，要描述整个网络，需要用三种概率分布来描述系统，即联合概率分布、边缘概率分布和条件概率分布。要搞清楚这三种不同的概率分布，是理解随机网络的关键。

第二，能量函数。随机神经网络是根植于统计力学的。受统计力学中能量泛

函的启发，引入了能量函数。能量函数是描述整个系统状态的一种测度。系统越有序或者概率分布越集中，系统的能量越小。反之，系统越无序或者概率分布越趋于均匀分布，则系统的能量越大。能量函数的最小值，对应于系统的最稳定状态。一般情况下，我们就是要找到系统最稳定的状态，也就是要找到能量函数最小值。

4.4.3 RBM 的基本概念

1. RBM 的结构

受限玻尔兹曼机包括隐层、可见层和偏置层。与前馈神经网络不一样，RBM 在可见层和隐层间的链接是方向不定的（值可以进行双向传播，也就是隐层→可见层和可见层→隐层）和完全链接的。

在标准的 RBM 中，隐含和可见层的神经元都是二进制表示［即神经元的激活值只能是服从伯努力分布（Bernoulli distribution）的 0 或 1］，不过也存在其他非线性的变种。

假设有一个二部图如图 4-9 所示，每一层的节点之间没有链接，一层是可视层，即输入数据层（v），一层是隐藏层（h），如果假设所有的节点都是随机二进制变量节点（只能取 0 或者 1 值），同时假设全概率分布 $p(v,h)$满足 Boltzmann 分布，我们称这个模型是 RBM。

以上的 RBM 网络结构有 n 个可视节点和 m 个隐藏节点，其中每个可视节点只和 m 个隐藏节点相关，其他可视节点是独立的，就是这个可视节点的状态只受 m 个隐藏节点的影响，对于每个隐藏节点也是，只受 n 个可视节点的影响，这个特点使得 RBM 的训练变得容易了。RBM 网络有几个参数，一个是可视层与隐藏层之间的权重矩阵 $\boldsymbol{W}m \times n$，一个是可视节点的偏移量 $b=(b_1,b_2,\cdots,b_n)$，一个是隐藏节点的偏移量 $c=(c_1,c_2,\cdots,c_m)$，这几个参数决定了 RBM 网络将一个 n 维的样本编码成一个什么样的 m 维的样本。

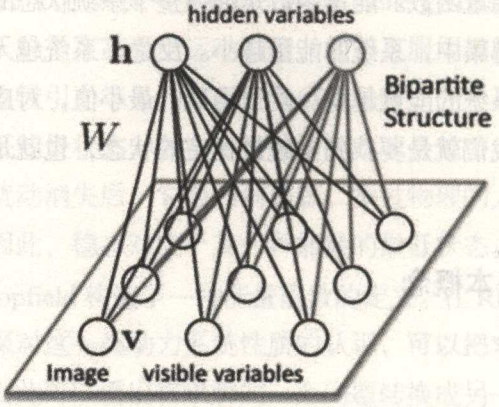


图 4-9　RBM 结构的可视节点和隐藏节点

2. RBM 的用途

那么 RBM 有什么用呢？RBM 的用途主要是两种，一种是对数据进行编码，然后交给监督学习方法进行分类或回归；另一种是得到了权重矩阵和偏移量，供 BP 神经网络初始化训练。

第一种可以说是把它当作一个降维的方法来使用。这种方式类似于稀疏自动编码器机理。

第二种用途比较奇怪。神经网络要训练一个权重矩阵和偏移量，但是如果直接用 BP 神经网络，初始值选得不好的话，往往会陷入局部极小值。简单解释下，就是你去跨栏，你的目标是让脚落到两个栏的中间，而如果有局部最小值限制，你的脚总是会踩在两个栏之上，而且来回计算，脚永远只能落在两个栏上，因为没有更小的值，让你能落到两个栏的中间。实际应用结果表明，直接把 RBM 训练得到的权重矩阵和偏移量作为 BP 神经网络初始值，得到的结果会非常好。

这就类似爬山。让你随便选座山来爬，要求你能爬上这座山最高的山峰，而你的体力最多只能爬上两个山峰，也不知道哪个山峰最高。这样，你就很容易爬到一个不是最高的山峰上，但如果用无人机拍一个全景图，你就知道哪个山峰最高，就能轻松地爬上那个山峰。这个时候，RBM 的角色就是那个无人机，而那个

山峰就是合适的初始权重。

其实，RBM 还有另外两种用途。

第三种，RBM 可以估计联合概率 $p(v,h)$，如果把 v 当作训练样本，h 当作类别标签（隐藏节点只有一个的情况，能得到一个隐藏节点取值为 1 的概率），就可以利用贝叶斯公式求 $p(h|v)$，就可以进行分类，类似朴素贝叶斯、LDA 和 HMM。说得专业点，RBM 可以作为一个生成模型（Generative Model）使用。

第四种，RBM 可以直接计算条件概率 $p(h|v)$，如果把 v 当作训练样本，把 h 当作类别标签（隐藏节点只有一个的情况，能得到一个隐藏节点取值为 1 的概率），RBM 就可以用来进行分类。说得专业点，RBM 可以作为一个判别模型（Discriminative Model）使用。

4.4.4 再看流行感冒的例子

为了说明对比差异，我们使用与上例相同的流感症状的数据集。测试网络是一个包含 6 个可见层神经元、2 个隐含层神经元的 RBM。我们用对比差异的方法对网络进行训练，将症状 v 赋到可见层中。在测试中，这些症状值被重新传到可见层；然后再被传到隐含层。隐含层的神经元表示健康/生病的状态，与自编码器相似。

在进行过几百次迭代后，我们得到了与自编码器相同的结果：输入一个生病样本，其中一个隐含层神经元具有更高激活值；输入健康的样本，则另一个神经元更兴奋。

到现在为止，我们已经学习了隐含层中强大的特征探测器——自编码器和 RBM，但现在还没有办法有效地利用这些功能。实际上，上面所用到的这些数据集都是特定的，而我们要找到一些方法来间接地使用这些探测出的特征。

好消息是，已经发现这些结构可以通过栈式叠加来实现深度网络。这些网络可以通过贪心法的思想训练，每次训练一层，以克服之前提到的在反向传播中梯度消失及过度拟合的问题。

这样的算法架构十分强大，可以产生很好的结果。如 Google 著名的“猫”识别，在实验中通过使用特定的深度自编码器，在无标记的图片库中学习到人和猫脸的识别。

4.5 DBN

和自编码器一样，我也可以将 RBM 像砖块一样叠加起来构建一个网络，这个网络就叫 DBN。

DBN 由多个 RBM 层组成，一个典型的 DBN 网络如图 4-10 所示。这些网络里只有一个可视层和一个隐层，层间互相连接，但层内的单元不会互相连接。每一层隐层的单元将训练成如何表示高级特征。

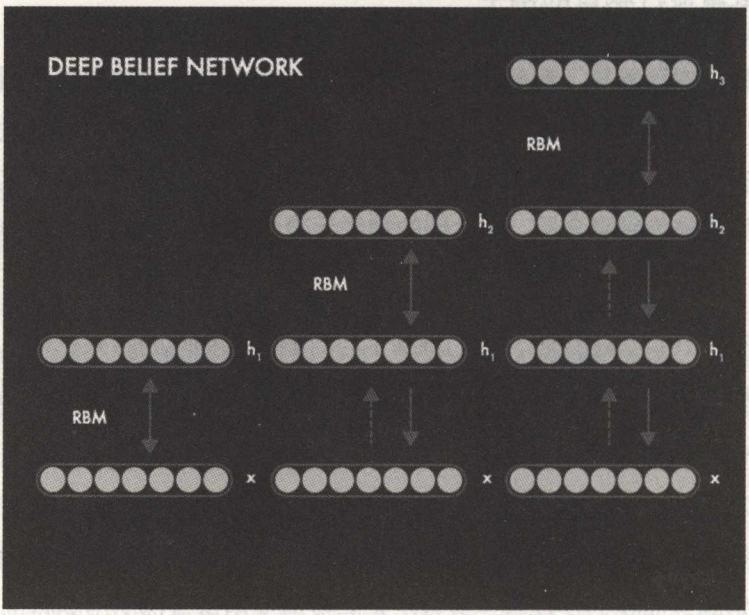


图 4-10 DBN 网络结构

首先，我们不考虑顶层构成联想记忆的两层，一个 DBN 的链接是通过从顶往下一层层生成的权值确定的。像之前所述，单个 RBM 就像一个砖块一样，它

更容易学习两两相连的权值。

辛顿在提出 DBN 时，验证了逐层贪婪训练方法(Greedy Layer-wise Training)，在大多数训练中非常有效，如图 4-11 所示。

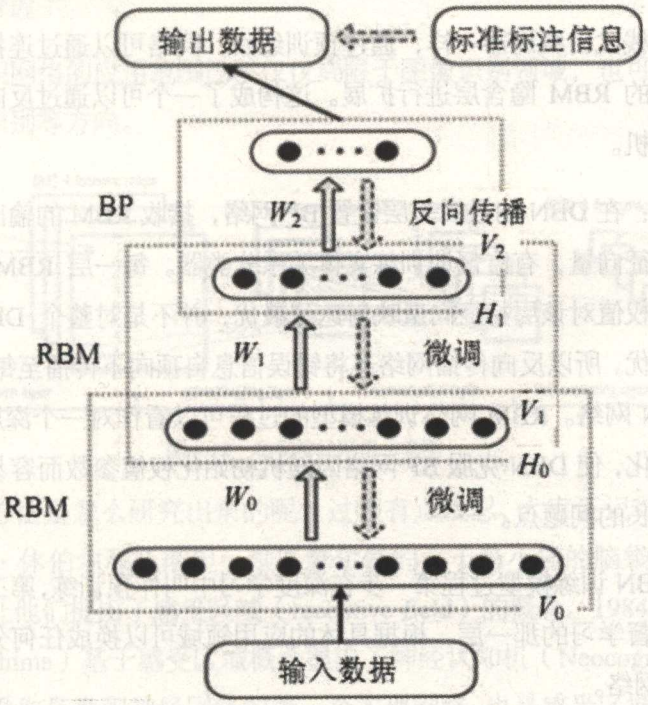


图 4-11 DBN 训练方法

DBN 在训练模型的过程中主要分为如下两步。

第一步：分别单独无监督地训练每一层 RBM 网络，确保特征向量映射到不同特征空间时，都尽可能多地保留特征信息。

在本例中，隐含层 RBM H_0 可以看作是 RBM H+1 的可见层。这里我们记作 $V_1 \cdots\cdots V_n$，第一个 RBM 的输入层即是整个网络的输入层，层间贪心式的预训练的工作模式如下。

(1) 通过对比差异法对所有训练样本训练第一个 RBM H_0。

（2）训练第二个 RBM H_1。由于 H_1 的可见层是 H_0 隐含层，训练开始于将数据赋至 V_0 可见层，通过前向传播的方法传至 H_0 隐含层。然后作为 H_1 的对比差异训练的初始数据。

（3）对所有层重复前面的过程。

（4）和栈式自编码器一样，通过预训练后，网络可以通过连接到一个或多个层间全连接的 RBM 隐含层进行扩展。这构成了一个可以通过反向传播进行微调的多层感知机。

第二步：在 DBN 的最后一层设置 BP 网络，接收 RBM 的输出特征向量作为它的输入特征向量，有监督地训练实体关系分类器。每一层 RBM 网络只能确保自身层内的权值对该层特征向量映射达到最优，并不是对整个 DBN 的特征向量映射达到最优，所以反向传播网络还将错误信息自顶向下传播至每一层 RBM，微调整个 DBN 网络。RBM 网络训练模型的过程可以看作对一个深层 BP 网络权值参数的初始化，使 DBN 克服 BP 网络因随机初始化权值参数而容易陷入局部最优和训练时间长的问题点。

上述 DBN 训练模型过程第一步在深度学习中叫作预训练，第二步叫作微调。最上面有监督学习的那一层，根据具体的应用领域可以换成任何分类器模型，而不必是 BP 网络。

本过程和栈式自编码器很相似，只是用 RBM 将自编码器进行替换，并用对比差异算法替换反向传播。

4.6 卷积神经网络

卷积神经网络是本章最后一个模型架构，如图 4-12 所示。卷积神经网络也被叫作 CNN，是人工神经网络的一种。它是一种特殊的对图像识别的方式，属于非常有效的带有前向反馈的网络。

CNN 诞生的主要目标是为识别二维图形，它的网络结构对平移、比例缩放、倾斜或其他形式的变形具有高度不变性。为啥会这样呢，因为每层关注的特征不一样，贴近原图的，关注的是像素级别的，而经过多次特征提取后，关联型、序列型或结构化等类型的特征（如拓扑结构）被提取出来，其一致性与事物本身的一致性就比较接近了。

现在，卷积网络的应用范围已不仅仅局限于图像识别领域，也可以应用在人脸识别、文字识别等方向。

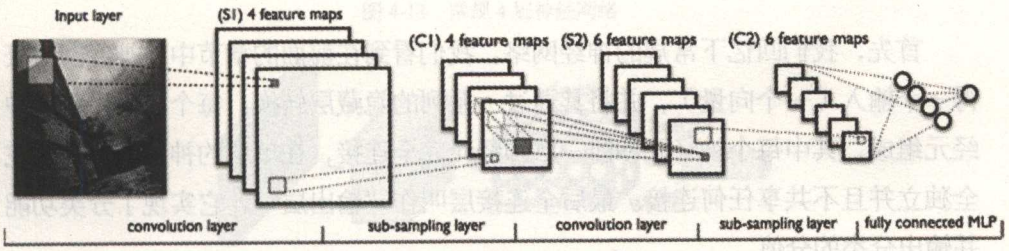


图 4-12　卷积神经网络示意图

这么牛的方法是怎么研究出来的呢？过程有点残忍，大家还记得 3.1 节中我们描述了大卫·休伯尔和托斯坦·维厄瑟尔做的关于给小猫的脑袋通电的实验吗？这个实验让他们提出了感受区域（receptive field）的概念。1984 年，日本学者福岛（Fukushima）基于感受区域概念提出了神经认知机（Neocognitron）。神经认知机可以看作是卷积神经网络的第一个实现网络，也是感受区域概念在人工神经网络领域的首次应用。神经认知机将一个视觉模式分解成许多子模式（特征），然后进入分层递阶式相连的特征平面进行处理，这样就可以将视觉系统模型化，使其能够在物体有位移或轻微变形的时候，也能完成识别。看出这个模式和前述模式中若干过滤器的相似之处了吧？是的，数学模拟就是借鉴了神经认知科学的发现。

通常，神经认知机包含两类神经元，即承担特征抽取的 *S*-元和抗变形的 *C*-元。*S*-元中涉及两个重要参数，即感受区域与阈值参数，感受区域确定输入连接的数目，阈值则控制对特征子模式的反应程度。每个 *S*-元的感光区中由 *C*-元带来

的视觉模糊量呈正态分布。也就是说，如果眼睛感受到物体是移动的，即已经感受到模糊和残影，*S*-元感光区会调整识别模式，这时，它不会完整地提取所有的特征给大脑，而会只获取一部分关键特征传给大脑，而屏蔽其他的视觉干扰。也就是说，眼睛在看到移动物体时，先由 *C*-元决定整体的特征感受控制度，再由 *S*-元感光区提取相应特征。为了有效地形成这种非正态模糊，福岛提出了带双 *C*-元层的改进型神经认知机。

4.6.1 卷积神经网络的结构

首先，我们回忆下常规的神经网络。我们看到在前面的章节中，神经网络获得一个输入（一个向量），并将其通过一系列的隐藏层转换。每个隐层由一组神经元组成，其中每个神经元和前一层神经元完全连接，在单层的神经元的功能完全独立并且不共享任何连接。最后全连接层叫作“输出层”，它实现了分类功能并输出分类的分值。

常规神经网络不能很好地适应所有的图像。在 CIFAR-10 训练集中，图片的大小只有 $32 \times 32 \times 3$（32 宽 32 高 3 颜色通道），所以一个全第一隐层的神经元常规神经网络将有 $32 \times 32 \times 3 = 3072$ 个。这个数字看起来可以接受，但显然这种全连通结构不能适应更大的图片。例如，一个图像的 respectible 大小为 $200 \times 200 \times 3$，会导致神经元有 $200 \times 200 \times 3 = 120000$ 个。此外，我们几乎肯定需要几个更少的神经元，让参数加起来快！显然，这种完全连接的结构很浪费，并且大量的参数会很快导致过度拟合。

三维结构的网络容量：卷积神经网络有很大的优势，对于包含巨量图片的输入而言，它以一种更合理的方式限制结构。特别是，不同于一般的神经网络，包含神经元的卷积网络的层安排在三个维度上：长、宽和高（注意，深度这个词在这里指的是三维空间的体积，而不是一个神经网络中层的数量）。例如，在 CIFAR-10 集合的图像是一个 $32 \times 32 \times 3$（分别是长、宽、颜色）。我们将看到，一层的神经元只能连接到它之前的层的小区域内，这替代了之前完全连接的那种方式。此外，CIFAR-10 图片格式最终的输出层维度是 $1 \times 1 \times 10$，在卷积网络的

最后，我们将完整图像减少到一个维度的分类分值，如图 4-13 和图 4-14 所示。

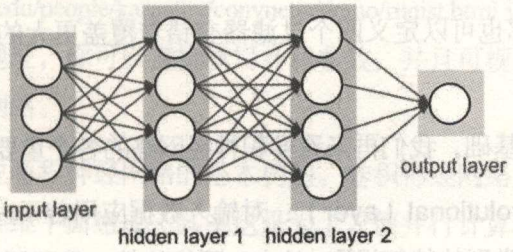


图 4-13 常规 4 层神经网络

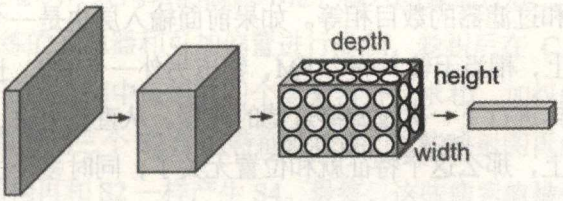


图 4-14 包含三维空间中神经元（宽度、高度、深度）的卷积神经网络，作为可视化的一个层

卷积网络的每一层都将三维输入转换成三维输出值。在本例中，输入层保留了图像的所有细节，中间三维向量图的两维表示了图像本来的宽度和高度，而深度表示了图像的颜色。

在我们深入了解实际的卷积神经网络之前，先简单了解卷积神经网络的每一层都在干什么。官方的说法：每一层会定义一个图像过滤器，或者称为一个有相关权重的方阵。这里其实就是特定特征提取，这个特定是根据函数或是人为设定或是其他情况来定的。举些不太恰当的例子，人类对人脸或脸型的东西会比较敏感，这类特征会优先提取出来；青蛙的眼睛对运动的物体很敏感（这个例子不适用于静止图片），会优先提取运动部分的特征；猫对耗子、猎手对猎物等很敏感；还有就像对特定的颜色和形状搭配很敏感，会提取特征出来，比如，好色的男士对某些特殊特征有很高的敏感度，你懂的。对于图片类的层，就是直线/折线、颜色、明暗、非连续性等特征，只要能表示为一种数学特征就行。每层的过滤器可以应用到整个图片上，也可以应用到一小部分图片上，所以通常每层（通常每层

的特征提取方式是相同的）可以有多个过滤器。什么叫过滤器？通常定义一个范围内的最小特征值就是过滤器，比如，你可以应用四个 6×6 的过滤器在一张 12×12 的图片上，你也可以定义四个过滤器交错着覆盖更大的范围（如 7×7/8×8/10×10）。

看完了上面的基础，我们再来看卷积神经网络的各个重要组成部分。

卷积层（Convolutional Layer）：对输入数据应用若干过滤器，一个输入参数被用来做了很多类型的特征提取。比如，图像的第一卷积层使用 4 个 6×6 过滤器，对图像应用一个过滤器之后得到的结果被称为特征图谱（Feature Map，FM），特征图谱的数目和过滤器的数目相等。如果前面输入层也是一个卷积层，那么过滤器应用在 FM 上，相当于输入一个 FM，输出另外一个 FM。也就是将滤波的特征值当成输入，再进行下一次过滤器过滤的过程。从直觉上讲，如果将一个权重分布到整个图像上，那么这个特征就和位置无关了，同时多个过滤器可以分别探测出不同的特征。比如，像青蛙一样，有的负责提取运动特征，有的负责提取蚊子特征，一旦两个特征都提取出来，就向特征重合的地方吐舌头。

子采样层（Subsample Layer）：又叫池化层（Pooling Layer），缩减输入数据的规模。例如，输入一个 12×12 的图像，并通过一个 6×6 的子采样，那么可以得到一个 2×2 的输出图像，这意味着原图像上的 36 个像素合并成为输出图像中的一个像素。实现子采样的方法有很多种，最常见的是最大值合并、平均值合并及随机合并。这其实类似于给一个图层打了马赛克，再对单个马赛克进行特征提取，对计算机来说，可以减少计算量。

最后一个子采样层（或卷积层）通常连接到一个或多个全连接层，全连接层的输出就是最终的输出。全连接层将计算分类的分值，得到一个[1×1×10]一维矩阵，也就是向量，其中的 10 对应一个分类的分值，如 CIFAR-10 训练集中的 10 个类别。顾名思义，与普通神经网络相同，这一层中的每个神经元都将和上一层的每个神经元连接。

训练过程通过改进的反向传播实现，将子采样层作为考虑的因素并基于所有

值来更新卷积过滤器的权重。实操中也可以设定前向反馈一些数据，以便调整。

http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html 这个链接是 MNIST Demo 在线例子的地址，你可以直接填写计算参数，并且可视化计算过程，以便更直观地理解卷积网络。

下面我们再说说卷积神经网络的基本构型。卷积神经网络是一个多层的神经网络，每层由多个二维平面组成（其实这是为了方便并行计算），而每个平面由多个独立的神经元组成。

图 4-15 所示是卷积神经网络的概念示范：输入图像通过和三个（实际是几个看实际情况）可训练的过滤器和可加偏置进行卷积，卷积后在 C1 层产生三个特征映射图，然后特征映射图中每组的四个像素再进行求和、加权值和加偏置，通过一个 sigmoid 函数得到三个 S2 层的特征映射图。这些映射图再经过过滤器得到 C3 层。这个层级结构再和 S2 一样产生 S4。最终，这些像素值被处理规则化，并连接成一个向量输入到传统的神经网络，得到输出。简单来说，就是去掉了读不懂的数据（符合一定特征，或者说可以提取特征的），留下了读得懂的数据。

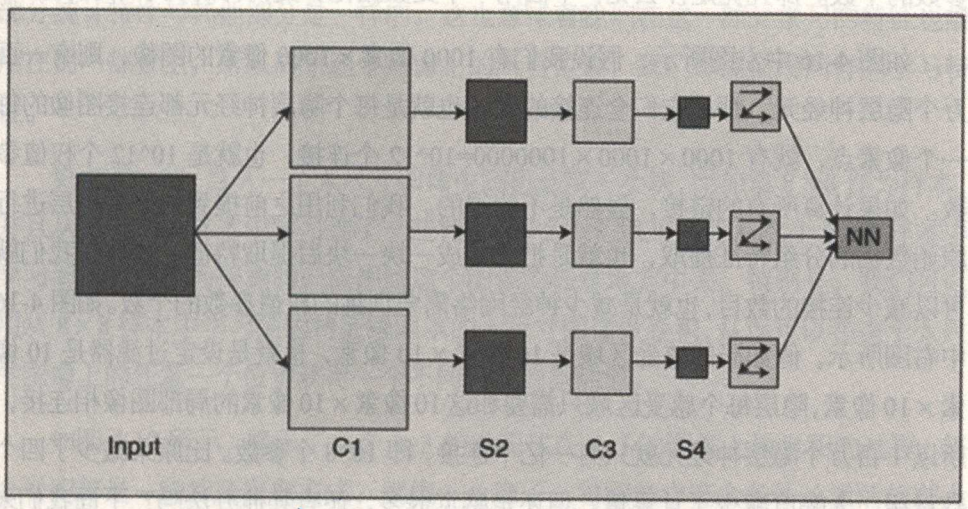


图 4-15 卷积网络基本构型

卷积神经网络的中间部分，就是真正在干卷积这件事的部分，是由两部分组

成的，一个是特征提取层，一个是特征映射层。C 层为特征提取层，每个神经元的输入与前一层的局部感受区域相连，并提取该局部的特征，一旦该局部特征被提取后，它与其他特征间的位置关系也随之确定；S 层是特征映射层，网络的每个计算层由多个特征映射组成，每个特征映射为一个平面，平面上所有神经元的权值相等。特征映射结构采用影响函数核小的 sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。

此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数，降低了网络参数选择的复杂度。卷积神经网络中的每一个特征提取层（C-层）都紧跟着一个用来求局部平均与二次提取的计算层（S-层），这种特有的两次特征提取结构使网络在识别时对输入样本有较高的畸变容忍能力。这个能力是以往的很多方式不支持的，也是多层卷积神经网络比较好用的原因之一。

4.6.2 关于参数减少与权值共享

CNN 牛的地方就在于通过感受区域和权值共享减少了神经网络需要训练的参数的个数。那究竟是什么呢？下面用一个更数据化，更具体的例子来说明。

如图 4-16 中左图所示，假设我们有 1000 像素 × 1000 像素的图像，则有一百万个隐层神经元，那么它们全连接的话，也就是每个隐层神经元都连接图像的每一个像素点，就有 1000 × 1000 × 1000000=10^12 个连接，也就是 10^12 个权值参数。如果计算所有的链接，显然是不值得的。我们利用之前提到的子采样层进行原始数据的分组特征提取，也就是把它分成一块一块后提取特征。这样，我们就可以减少连接的数目，也就是减少神经网络需要训练的权值参数的个数。如图 4-16 中右图所示，假如局部感受区域是 10 像素 × 10 像素，也就是设定过滤器是 10 像素 × 10 像素，隐层每个感受区域只需要和这 10 像素 × 10 像素的局部图像相连接，所以 1 百万个隐层神经元就只有一亿个连接，即 10^8 个参数。比原来减少了四个数量级，大幅度减少了计算量，但还是感觉很多，还有其他办法吗？下面我们来看看第二个神器——权值共享。

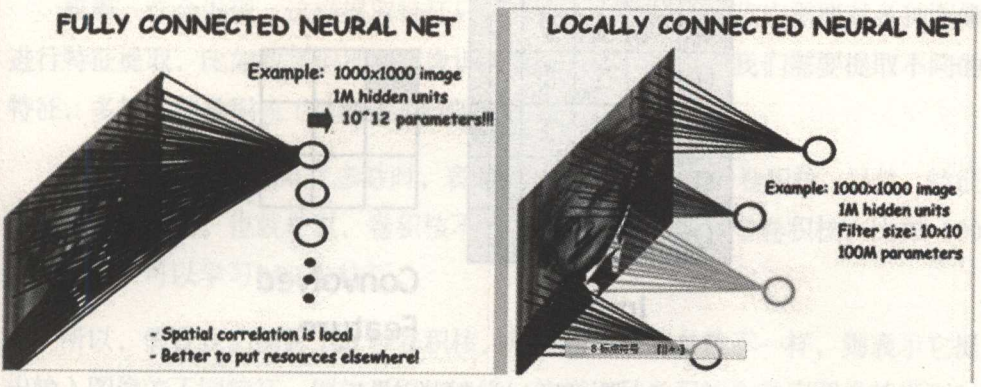


图 4-16 卷积网络减少计算量

在上面的局部连接中，每个神经元都对应 100 个参数，一共 1000000 个神经元，如果这 1000000 个神经元的 100 个参数都是相等的，那么参数数目就变为 100 了。

怎么理解权值共享呢？我们可以将这 100 个参数（也就是卷积操作）看成是提取特征的方式，该方式与图像上的位置无关。这其中隐含的原理是：图像的一部分统计特性与其他部分是一样的。这也意味着我们在这一部分学习的特征也能用在另一部分上，所以对于这个图像上的所有位置，我们都能使用同样的学习特征。

更直观一些，当从一个大尺寸图像中随机选取一小块，比如说 8×8 作为样本，并且从这个小块样本中学习到了一些特征，那么我们就可以把从这个 8×8 样本中学习到的特征作为探测器，应用到这个图像的任意地方中去。特别是，我们可以用从 8×8 样本中学习到的特征跟原本的大尺寸图像做卷积，从而可以在这个大尺寸图像上的任意一个位置获得一个不同特征的激活值。

如图 4-17 所示，展示了一个 3×3 的卷积核在 5×5 的图像上做卷积的过程。每个卷积都是一种特征提取方式，就像一个筛子，将图像中符合条件（激活值越大越符合条件）的部分筛选出来。

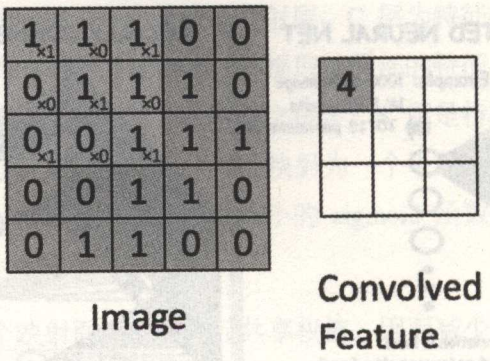


图 4-17（A） 卷积的过程 1

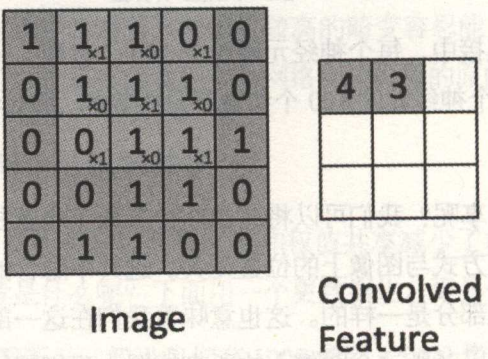


图 4-17（B） 卷积的过程 2

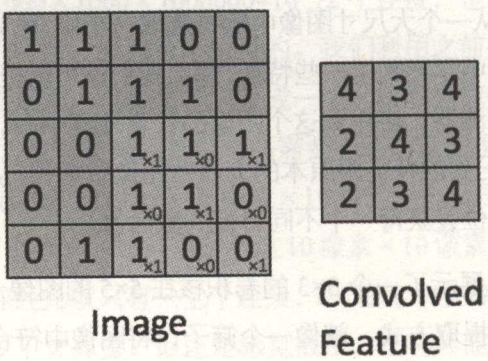


图 4-17（C） 卷积的过程 3

接着，你会发现，这样提取特征只是提取了一种特征。我们需要对各种图像进行特征提取，比如数字识别和图像识别就不一样。那么，我们需要提取不同的特征，多加几种卷积核（过滤器）行吗？

如上所述，只有 100 个参数时，表明只有 1 个 10×10 的卷积核，显然，特征提取是不充分的。也就是说，卷积核不够，我们可以添加多个卷积核，比如 100 种卷积核，可以学习 100 种特征。

所以，假设我们加到 100 种卷积核，每种卷积核的参数不一样，则表示它提出输入图像的不同特征，例如不同的边缘。这样每种卷积核去卷积图像就得到对图像的不同特征的表示，我们称之为 FM。所以 1 种卷积核就有 100 个 FM。这样我们就能很清晰地得到我们这一层的参数是：100 种卷积核×每种卷积核共享 100 个参数=100×100=10000，也就是 1 万个参数。看一下最开始我们的参数是 10^12，而且只是一种特征提取，现在我们把它缩减到 1 万个参数，还能提取 100 种特征，这相当于利用算法的进步降低了很多计算的成本！

如图 4-18 中左图所示，不同的颜色表达不同的卷积核。

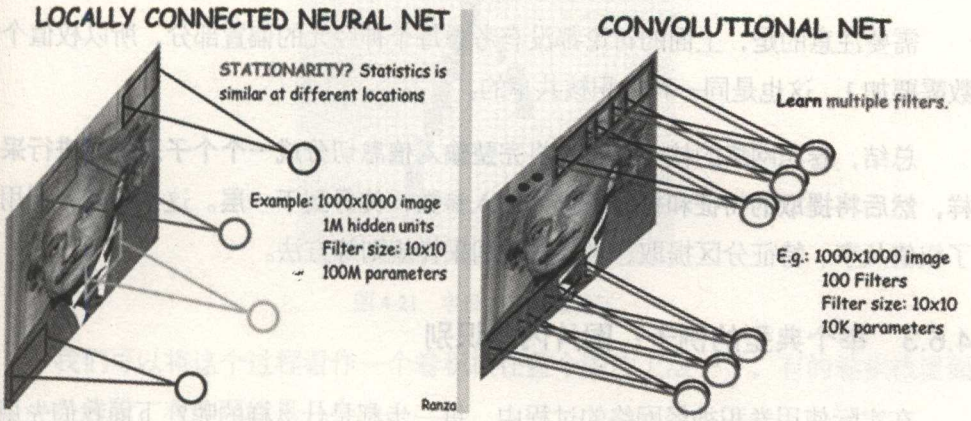


图 4-18　卷积网络参数缩减

刚才说隐层的参数个数和隐层的神经元个数无关，只和卷积核的大小和卷积核种类的多少有关。那么，隐层的神经元个数我们如何确定呢？它和原图像输入单元大小（神经元个数）、卷积核的大小和卷积核在图像中的滑动步长都有关！

那什么是滑动步长呢？步长就是决定卷积核覆盖范围的参数。例如，图像是 1000 像素×1000 像素，卷积核大小是 10 像素×10 像素，假设卷积核没有重叠，它的步长定义为 10，那么隐层的神经元个数就是(1000×1000)/ (10×10)=100×100 个神经元。假设步长是 8，即卷积核会重叠两个像素，那么……不算了，懂思路就好。注意，这只是一种卷积核，也就是一个 FM 的神经元个数，如果 100 个 FM 就是 100 倍了。由此可见，图像越大，神经元个数和需要训练的权值参数个数的差距就越大，如图 4-19 所示。

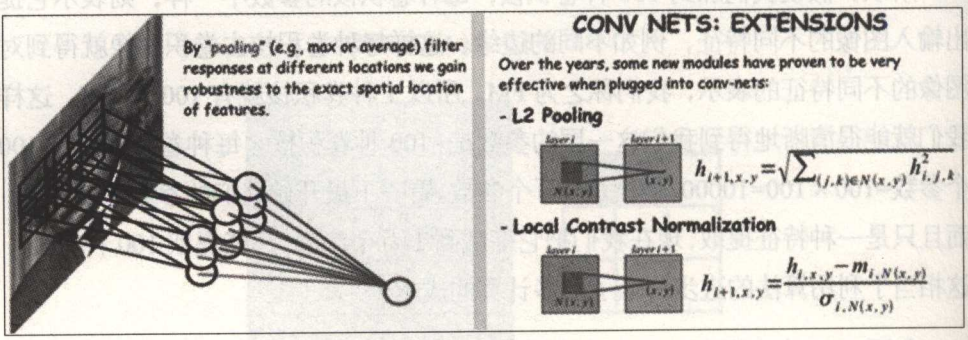


图 4-19　权值参数计算

需要注意的是，上面的讨论都没有考虑每个神经元的偏置部分，所以权值个数需要加 1，这也是同一种卷积核共享的。

总结，卷积网络的核心思想是将**完整输入信息切分成一个个子采样层进行采样，然后将提取的特征和权重值作为输入参数，传导到下一层**。这其中充分利用了权值共享、特征分区提取、时间或空间采样规则等方法。

4.6.3　举个典型的例子：图片内容识别

在实际使用卷积神经网络的过程中，每一步都是什么样的呢？下面我们先用一个字母 A 的识别过程举例。

首先，像素化，然后采样，识别各类特征。图 4-20 所示从下往上数第二根和第三根线对应的像素点就是 2 个竖着的像素向左平移，并且颜色都变化的特征。

我们把所有的点都往左平移，得到了一个近似于“A”的图像，识别出来就是类似图 4-21 所示的结果。

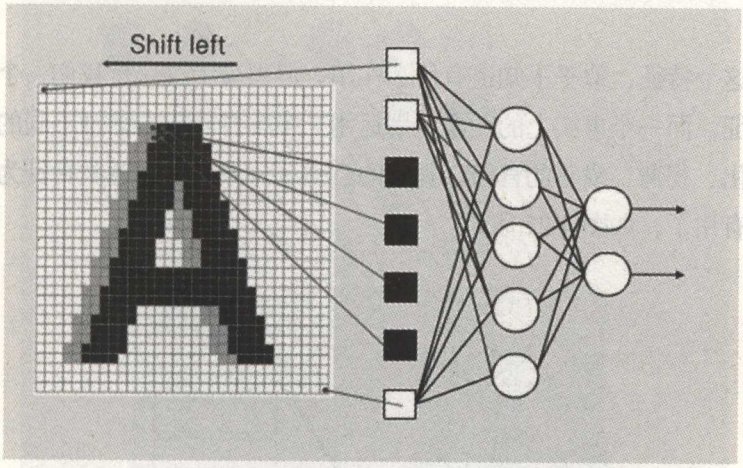


图 4-20 字母 A 的识别

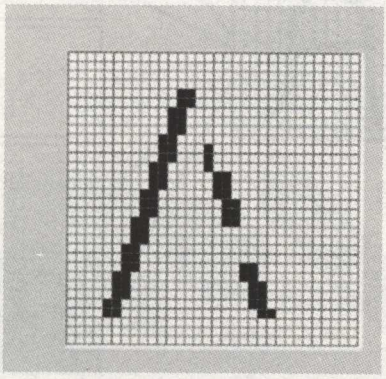


图 4-21 字母 A 的像素特征

我们可以将这个过程看作一个卷积核在整个图片上滚一下，有的卷积核里面是一条线段，有的可能是几个点。

开始对“A”的采样的主要目的是混淆特征的具体位置，我们找出某一个一般性特征后，这个特征所处的具体位置都不重要了。比如，这个“A”，如果我们识别了上半部分，得到一个“A”，不用知道它的具体位置，只用知道它下面

如果有两斜杠“”“”，就可以识别出这是个“A”，“A”上下左右移动变形都不影响我们识别它。这种方式天然地对于变形扭曲（手写不规则）图片有较好的识别率。

单看这个特征，似乎不知道有什么作用，但别忘了这只是我们一个卷积核学习到的特征，而一个事实上的图像识别运用了多个卷积核去识别不同的特征，比如颜色变化、拐弯、横着的直线、竖着的直线……然后，以这类特征为基础进行判断就会有用了，如图 4-22 所示。

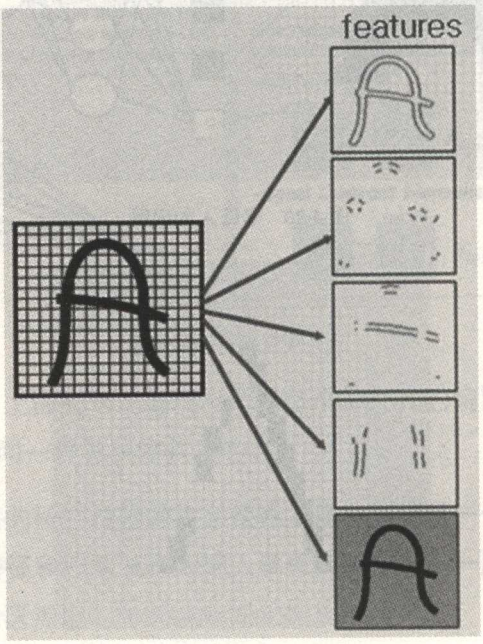


图 4-22　特征提取示意图

根据卷积核学习到的特征去映射，也就是说，第一次特征找完了之后，接着将第一层发现的特征作为第二层的输入，做第二次的特征查找，也就是我们把得到的特征更抽象化出来，比如拐弯和直线之间的连接这种类型的特征，如图 4-23 所示。当然，你也可以用不同的 FM 组合得到不同的 FM，有点像最近谷歌出的用机器生成“未知生物”图片。

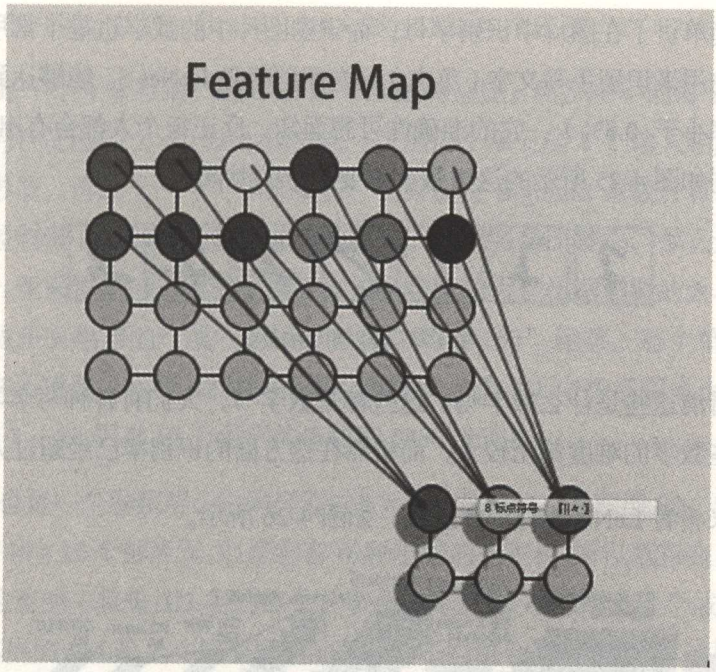


图 4-23 特征映射

然后，我们就可以用实验证明分类识别的效果了，如图 4-24 所示。

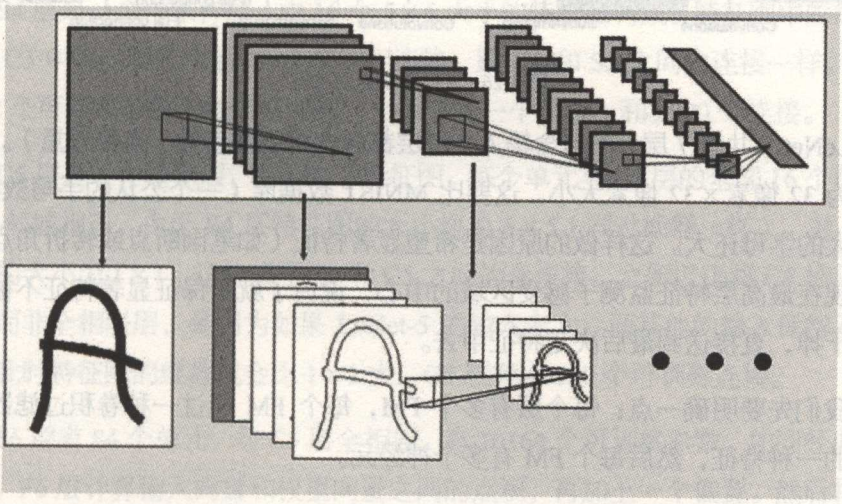


图 4-24 分类识别效果

上面简单讲了在图片中识别字母，而识别图片中的数字也是个常用的应用，一种典型的用来识别手写文字（英文）的卷积网络是 LeNet-5。能够达到商用的地步（错误率小于 0.8%），它的准确性可想而知。反正每个人都会有出错的时候，尤其是遇到如图 4-25 所示的这类数字需要区分的时候。

图 4-25　难以辨认的数字

你能看清这些是什么数字吗？这些都是数字 7，人们有各种写字习惯，靠肉眼识别这些数字的难度都比较大，而机器在这方面的识别率已经超过人类。

我们来看看 LeNet-5 的识别过程，如图 4-26 所示。

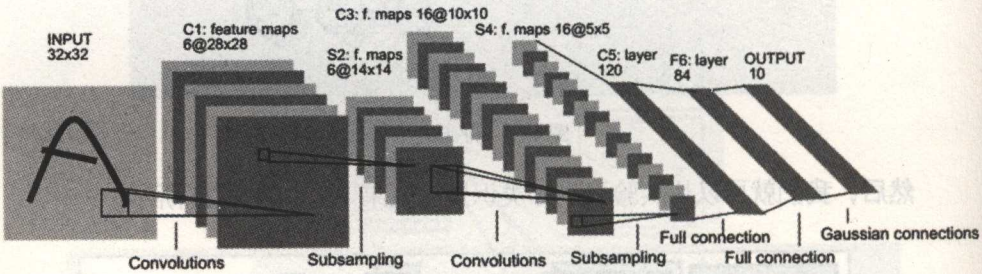


图 4-26　A 的卷积识别过程

LeNet-5 共有 7 层，不包含输入，每层都包含可训练参数（连接权重）。输入图像为 32 像素 × 32 像素大小。这要比 MNIST 数据库（一个公认的手写数据库）中最大的字母还大。这样做的原因是希望显著特征（如笔画断点或转折角点）能够出现在最高层特征监测子感受区域的中心，说白了就是保证显著特征不被每次迭代干掉，直接送到最后决定特征中去。

我们先要明确一点：每个层有多个 FM，每个 FM 通过一种卷积过滤器提取输入的一种特征，然后每个 FM 有多个神经元。

C1 层是一个卷积层，由 6 个特征图 FM 构成。特征图中每个神经元与输入中 5 × 5 的邻域相连。特征图的大小为 28 × 28，这样能防止输入的连接掉到边界之外。

C1 有 156 个可训练参数，共 156×(28×28)=122304 个连接。

S2 层是一个子采样层，有 6 个 14×14 的特征图。特征图中的每个单元与 C1 中相对应的特征图的 2×2 邻域相连接。S2 层每个单元的 4 个输入相加，乘以一个可训练参数，再加上一个可训练偏置，结果通过 Sigmoid 函数计算。可训练系数和偏置控制着 Sigmoid 函数的非线性程度。如果系数比较小，那么运算近似于线性运算，亚采样相当于模糊图像。如果系数比较大，根据偏置的大小，亚采样可以被看成是有噪声的“或”运算或者有噪声的“与”运算。每个单元的 2×2 感受区域并不重叠，因此 S2 中每个特征图的大小是 C1 中特征图大小的 1/4（行和列各 1/2）。S2 层有 12 个可训练参数和 5880 个连接。

C3 层也是一个卷积层，它同样通过 5×5 的卷积核去卷积层 S2，然后得到的 FM 就只有 10×10 个神经元，但是它有 16 种不同的卷积核，所以就存在 16 个 FM。这里需要注意的一点是：C3 中的每个 FM 是连接到 S2 中的所有 6 个或者几个 FM 的，表示本层的 FM 是上一层提取到的 FM 的不同组合（看到没有，这里是组合，就像之前聊到的人的视觉系统一样，底层的结构构成上层更抽象的结构，例如，边缘构成形状或者目标的部分）。

S4 层是一个子采样层，由 16 个 5×5 大小的特征图构成。特征图中的每个单元与 C3 中相应特征图的 2×2 邻域相连接，跟 C1 和 S2 之间的连接一样。S4 层有 32 个可训练参数（每个特征图一个因子和一个偏置）和 2000 个连接。

C5 层是一个卷积层，有 120 个特征图。每个单元与 S4 层的全部 16 个单元的 5×5 邻域相连。由于 S4 层特征图的大小也为 5×5（同过滤器一样），故 C5 特征图的大小为 1×1，这构成了 S4 和 C5 之间的全连接。之所以仍将 C5 标示为卷积层而非全相联层，是因为如果 LeNet-5 的输入变大，而其他的输入保持不变，那么此时特征图的维数就会比 1×1 大。C5 层有 48120 个可训练连接。

F6 层有 84 个单元，与 C5 层全相连。有 10164 个可训练参数。如同经典神经网络，F6 层计算输入向量和权重向量之间的点积，再加上一个偏置。然后将其传递给 Sigmoid 函数产生单元 i 的一个状态。

最后，输出层由欧式径向基函数（Euclidean Radial Basis Function）单元组成，每类一个单元，每个有 84 个输入。换句话说，每个输出 ERBF 单元计算输入向量和参数向量之间的欧式距离。输入离参数向量越远，ERBF 输出就越大。一个 ERBF 输出可以被理解为衡量输入模式和与 ERBF 相关联类的一个模型的匹配程度的惩罚项。用概率术语来说，即模式的期望分类足够接近。

ERBF 参数向量起着 F6 层目标向量的角色。需要指出这些向量的成分是+1 或-1，这正好在 F6 Sigmoid 的范围内，因此，可以防止 Sigmoid 函数饱和。实际上，+1 和-1 是 Sigmoid 函数的最大弯曲的点处。这使得 F6 单元运行在最大非线性范围内。

简而言之，通过噪音过滤和特征提取，强化出真正有用的笔画（线条）拓扑关系，以此为基础，识别字母。通过大量的训练，能把不同字母的各种区分特征识别出来，只要足以区分不同字母，挑出非字母，就可以实现识别。幸好没来识别中文，否则会很难看。

讲了这么多，你能感受到卷积神经网络 CNN 具有哪些优势吗？CNN 主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于 CNN 的特征检测层通过训练数据进行学习，在使用 CNN 时，避免了显式的特征抽取，而隐式地从训练数据中进行学习；再者，由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，避免了特征提取和分类过程中数据重建的复杂度。

在很多情况下，图像显式的特征提取并不容易，在一些应用问题中也并非总是可靠的。卷积神经网络避免了显式的特征取样，隐式地从训练数据中进行学习。这使得卷积神经网络明显有别于其他基于神经网络的分类器，通过结构重组和减少权值将特征提取功能融合进多层感知器。它可以直接处理灰度图片，能够直接用于处理基于图像的分类。

卷积网络较一般神经网络在图像处理方面有如下优点。

（1）输入图像和网络的拓扑结构能更好地吻合。

（2）特征提取和模式分类同时进行，并同时在训练中产生。

（3）权重共享可以减少网络的训练参数，使神经网络结构变得更简单，适应性更强。

卷积神经网络目前已成为语音分析和图像识别领域的研究热点。它的权值共享网络结构使它更类似于生物神经网络，降低了网络模型的复杂度，减少了权值的数量。卷积神经网络的这个优点在网络的输入是多维图像时表现得更为明显，使图像可以直接作为网络的输入，避免了传统识别算法中复杂的特征提取和数据重建过程。

4.7 不会忘记你：循环神经网络

4.7.1 什么是 RNN

首先，要分清楚一个概念，RNN 是两种神经网络的缩写，一种是递归神经网络(Recursive Neural Network)，一种是循环神经网络(Recurrent Neural Network)，虽然这两个概念有千丝万缕的联系，但本书主要讨论的是第二种，也就是循环神经网络及其变种。

循环神经网络是指一个随着时间的推移，重复发生的结构。例如，如果你有一个序列 X = ['H', 'E', 'L', 'L']，该序列被送到一个神经元，而这个神经元的输出连接到它的输入上。

在步骤 0 的这个时刻，字母“H”是作为输入传入的，在步骤 2 时，字母“E”被作为输入传入。随着时间的推移展开这个网络，将变成图 4-27 所示的网络结构。

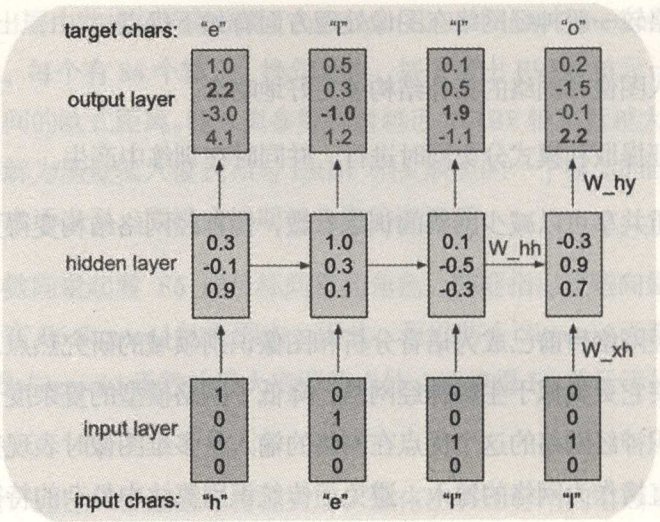


图 4-27　RNN 单元的展开

递归神经网络仅仅是广义化的循环神经网络。循环神经网络在一个序列的长度上的权重是共享的（并且维度保持不变）。因为，当遇到一个训练时间和测试时间长度不同的序列时，是不能处理位置独立权重的。递归网络的权重（与维数保持恒定）出于同样的原因在每一个节点被共享。

这意味着，所有的 W_xh 权重是相等的（共享），以及 W_hh 权重也是相等的。它是单个的神经元，并且能够及时展开。

递归神经网络看起来如图 4-28 所示。

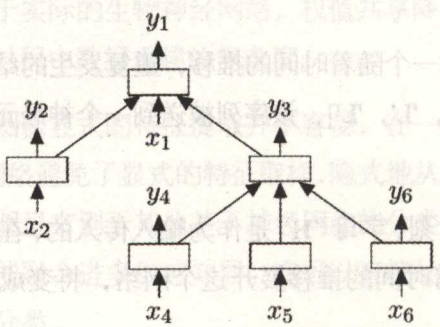


图 4-28　递归神经网络示意图

为什么被称为递归神经网络呢？因为每个父节点的子节点和另外一个节点类似。

到底使用哪种神经网络取决于实际情况。在 Karpathy 的博客中可以看到，他的项目是生成一个个字符，而处理中不需要分层，对这种情况来说循环神经网络是不错的选择。

如果你想生成一个解析树，用递归神经网络会更好些，因为它有助于创造更好的分层表示。

1. 循环神经网络

正式进入循环神经网络之前，来想一下我们的思考步骤，或者叫思考时序。

我们不会每一秒钟都从头开始思考。比如，当你看一本书时，会根据以往学习的知识理解每一个词。你会从上下文中产生联想，帮助你更好地理解这篇文章。当你冒出来一个想法或问题后，会通过读本书来归纳总结，试着印证你的想法或者回答你的问题。

人类的这一大特点，无法在传统的神经网络中找到类似的，这也是一般神经网络的一个缺点。例如，假设你要将电影中每个时刻发生的事按时间归类，传统的神经网络目前还无法做到，因为这需要使用之前在电影中出现的事件推理出后面发生的事情，而循环神经网络可以解决这一问题。它们可以在网络中循环，并能够维持信息，如图 4-29 所示。

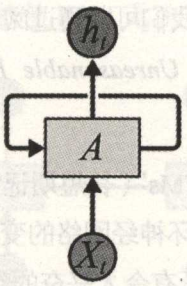


图 4-29 一个循环神经单元

1）循环神经网络有回路

在图 4-29 中，神经网络的单元 A，它的输入值是 x_t，输出值是 h_t。信息通过回路从网络的目前状态传递到下一个状态。同一个单元不停地处理不同的输入值，而这些值是自己产生的。大家觉得循环神经网络的回路很神秘吗？如果你深入思考会发现，它们与正常的神经网络没什么不同。反复出现的神经网络可以被认为是在同一个网络中的多个副本，每个都传递消息给继承者，也就是下个时态的神经元。试想，如果如图 4-30 所示展开循环会发生什么？

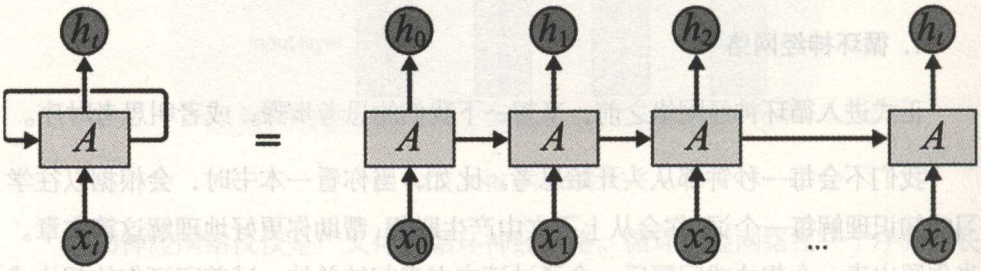


图 4-30　单元展开循环

2）已展开的递归神经网络

这个链式结构揭示了与循环神经网络密切相关的序列。它们是让神经网络能使用这些数据的一种自然结构。

这么好的东西肯定会被用在各个方面。在过去的几年里，RNN 已经成功地应用在各种问题上，并取得令人难以置信的成功。例如，语音识别、语言建模、翻译、字幕、图像……不胜枚举。我们可以通过阅读 Karpathy 的博客文章《循环神经网络不切实际的效果》（*The Unreasonable Effectiveness of Recurrent Neural Networks*）深入了解。

这个结果的关键是使用“LSTMs”（长短期记忆网络，Long Short Term Memory Networks），它是一种特殊的循环神经网络的变种，对于许多任务来说，这种方法比标准的 RNN 好得多。几乎所有令人兴奋的结果都是基于 LSTMs 实现的。本书后面章节将进一步探讨这些结构。

3）长时间依赖的问题

RNN 的诉求之一是，它也许能将以前的信息连接到当前任务。例如，我们有时需要使用前一个视频帧理解当前帧的内容。如果 RNN 能做到这一点，它们会非常有用，但它们能做到吗？视情况而定。

有时候，我们只需要看最近的信息来执行现在的任务。举个例子，考虑一个语言模型试图预测基于当前的下一个词。如果我们试图预测“天空中有_”这句话的最后一个字，那么我们不需要任何进一步的语境就可以判断下一个字是云或鸟。在这种情况下，如果相关的信息（这里指的是“天空中有”）和我们需要填词的位置之间的差距较小，那么 RNN 就能学会利用过去的信息，如图 4-31 所示。

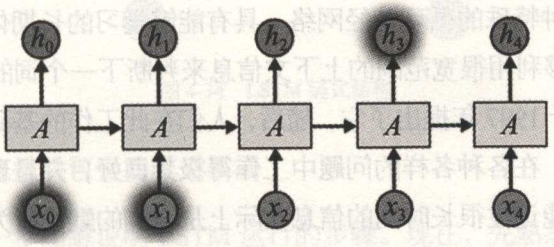


图 4-31 利用上下文知识

但有时，我们需要更多的上下文。试着预测“我在中国长大……（省略 20 个字），我讲一口流利的__。”的最后一个词。最近的信息表明，下一个字可能是语言的名字，但如果我们想要缩小语言名字的范围，则需要这个词的上下文。我们发现，有时相关上下文信息和我们需要得到的词的这个位置相差很大。

不幸的是，这种距离的增长将使 RNN 无法学习到这些信息，如图 4-32 所示。

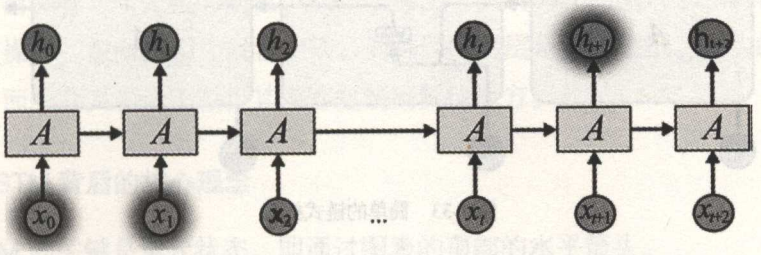


图 4-32 距离较长，无法利用上下文知识

我们仔细看图 4-32，h_{t+1} 需要 x_0 x_1 位置的信息，但由于距离较长，x_0 x_1 信息无法传导过来。

从模型结构上看，RNN 完全能够处理这样的“长期依赖”问题。理论上讲，人们可以仔细挑选参数，为它们解决这种小问题。但在实践中，我们发现 RNN 似乎并不能处理好这些问题。Hochreiter 和 Bengio 深入探讨了这一问题，发现了普通的 RNN 结构无法处理好这种问题。

值得庆幸的是，LSTMs 没有这个问题。

4.7.2 LSTM 网络

LSTM 是一种特殊的循环神经网络，具有能够学习的长期依赖的能力，比如在文本处理中能够利用很宽范围的上下文信息来判断下一个词的概率。Hochreiter 和 Schmidhuber 于 1997 年提出了它，随后，人们在此工作的基础上进行了很多完善和推广。LSTM 在各种各样的问题中工作得极其良好，并且现在正在被更广泛的使用。LSTM 能记住很长时间的信息实际上是它们的默认行为，而不是需要努力学习的东西。换句话说，这能力是打娘胎里带出来的。

所有的 RNN 都有神经网络的重复模块组成的链式结构。对于标准的 RNN，这种重复模块有一个非常简单的结构，如一个单一的 tanh（**双曲正切**）层，如图 4-33 所示。

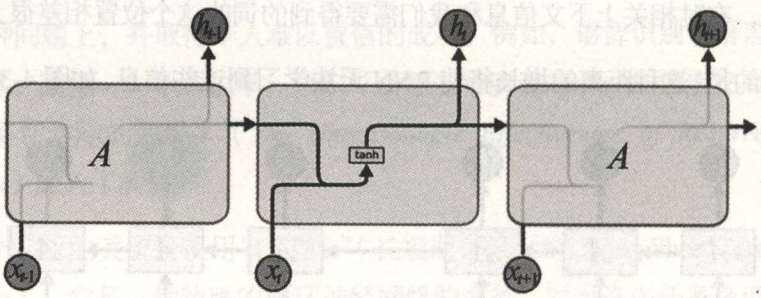


图 4-33　简单的链式结构

1. RNN 中包含单个层重复模块

LSTM 也有这样的链式结构，但重复模块却具有和一般 RNN 不同的结构。代替单个神经网络层的有四个，它们用一种特殊的方式进行交互，如图 4-34 所示。

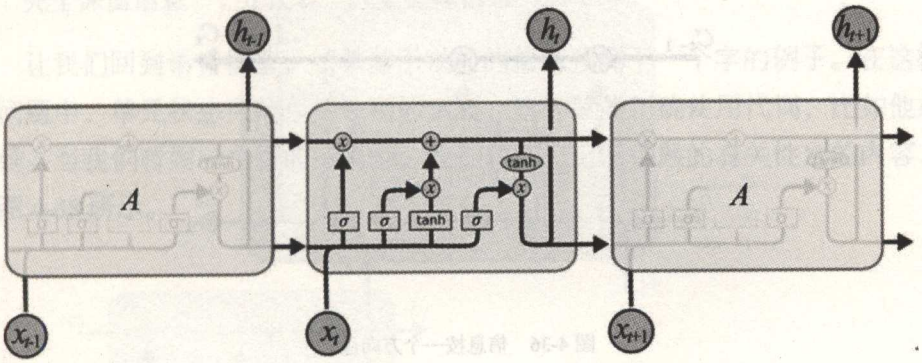


图 4-34 LSTM 链式结构

2. LSTM 重复模块包含四个交互层

我们会一步一步图解说明 LSTM 运行的步骤。现在，先熟悉下将要使用的符号，如图 4-35 所示。

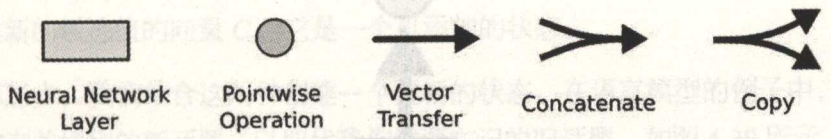


图 4-35 所使用的符号

每个箭头有一个完整的向量，从一个输出节点到另外的输入节点。粉红圆圈代表逐点操作，像向量相加或者相乘，而神经网络层是黄色的框。箭头的合并表示串联，而线分叉表示正在将其内容复制到其他地方。

3. LSTM 背后的核心理念

LSTM 的关键是单元状态，即通过图表的顶部的水平箭头。

单元状态有点像一个传送带。它通过整个链往下运行，只有一些小的线性相互作用。信息会很容易地沿着箭头方向流动，如图 4-36 所示。

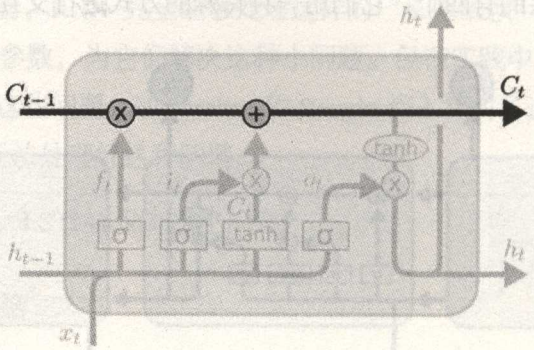


图 4-36　信息按一个方向运行

LSTM 完全可以删除或添加单元状态的信息，被称为门限（gates）的结构将会控制信息。

门限可以有选择地让信息通过，它由 Sigmoid 神经网络层和点乘操作组成，如图 4-37 所示。

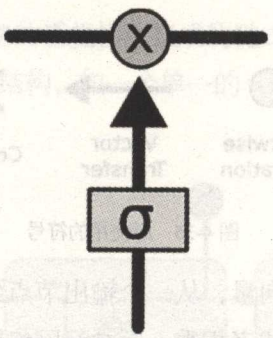


图 4-37　有选择的筛选信息

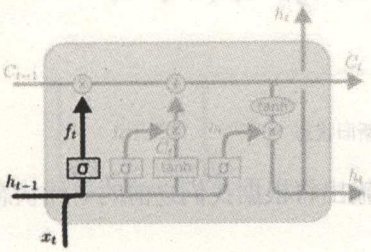
Sigmoid 层输出 0 和 1 之间的数字，描述每个组件能通过多少信息。0 值表示“不让任何东西通过”，而 1 值的意思是“让所有的都通过”。

一个 LSTM 有三种这样的门限来保护和控制单元的状态。

4. LSTM 分步执行

第一步，决定从单元状态中扔掉哪些信息。一个叫“遗忘门限”的 Sigmoid 层做出这些决定。在单元状态 c_{t-1} 上，h_{t-1} 和 x_t 输出 0 和 1 之间的一个数字。1 代表“完全保留信息”，0 代表“完全丢掉信息”。

让我们回到语言模型，试着基于以前的信息预测下一个字的例子。在这样的问题中，单元状态可能包括性别的话题，这样就能正确使用代词，比如他或者她。当我们看到一个新的话题时，我们想要忘记旧话题的有关性别的内容，如图 4-38 所示。

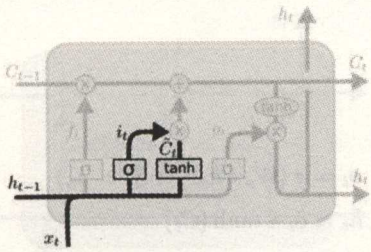


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

图 4-38 忘记旧话题

第二步，我们要决定在单元状态中储存哪些新的信息。这分为两个部分，首先，被称为“输入门限”的一个 Sigmoid 层决定哪些值会更新。接着，一个 tanh 层创建新的候选值的向量 C_t，它是一个可添加的状态。

第三步，我们结合这两种创建一个更新的状态。在语言模型的例子中，我们想添加有关性别的新话题，以取代我们需要忘记的旧话题，如图 4-39 所示。



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

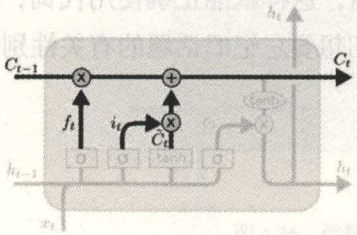
图 4-39 更新状态

再来更新旧单元状态 C_{t-1}，进入新的单元状态 C_t。前面的步骤已经说明了要

怎么做，现在我们只需要真正实现它。

我们乘以旧状态 f_t，丢弃我们之前决定忘记的东西。然后，我们加上 $i_t\tilde{C}$。这是新的候选值，由我们决定更新每个状态值。

在语言模型下，我们实际上把旧性别主题信息丢弃了，并添加了新的信息，如图 4-40 所示。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

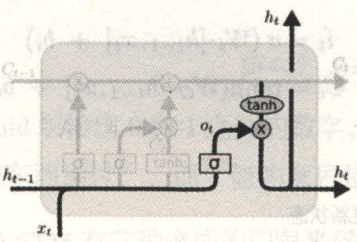
图 4-40　更新旧状态

最后，我们需要决定要输出什么。此输出将根据我们之前的单元状态，但这是经过筛选的版本。

来看看输出步骤：首先，我们运行一个 Sigmoid 层，它决定我们要输出哪些单元状态。

然后，我们把单元状态通过 tanh 函数（将输出值规一化于–1 到 1 之间）和 Sigmoid 门限的输出相乘，以便只输出我们决定输出的部分。

因为语言模型的例子只有一个主题，所以可能会输出一个有关动词的信息。例如，它可以输出的主语是单数还是复数，比如他或他们，然后就能确定接下来形成一个什么动词，如图 4-41 所示。



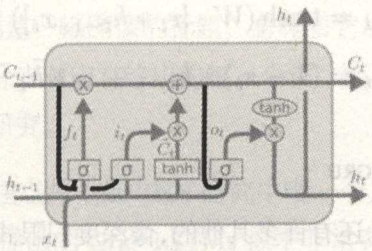
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

图 4-41　输出

4.7.3 LSTM 变体

到目前为止，描述的是一个非常正常的 LSTM，但并非所有 LSTM 都与上述相同。事实上，几乎每个涉及 LSTM 的论文都会使用略有不同的版本。差异是次要的，但其中一些值得拿出来说说。

一种流行的 LSTM 变体，由 Gers 和 Schmidhuber 引入，增加了观察口连接（Peephole Connections）。这意味着我们将使用门限层考察单元状态，如图 4-42 所示。

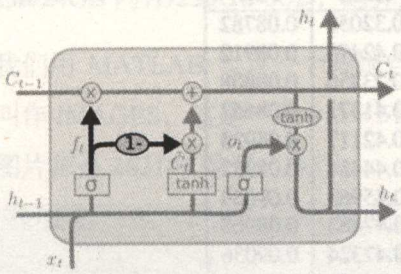


$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

图 4-42 观察口连接

所有门限增加了观察口，但许多论文认为是一部分门限有观察口，一部分没有。

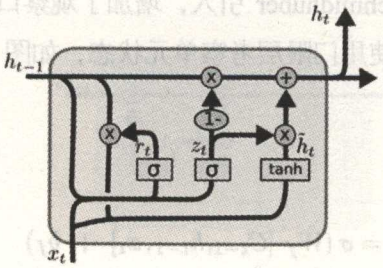
另一个变化是使用组队遗忘（也叫耦合忘记）和输入门（Coupled Forget and Input Gates）。它们的差别在于不是分别决定忘记什么或我们应该添加什么新信息，而是一起做出这些决定。我们只有在其位置输入数据时才会忘记。我们只有在忘记旧的数据时，才会输入新的值，如图 4-43 所示。



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

图 4-43 组队遗忘

LSTM 稍微戏剧性的变化是门限重复单元（the Gated Recurrent Unit），或称为 GRU，由 Kyunghyun Cho 等人于 2014 年引入。它结合了遗忘和输入门限，合成了一个“更新门限”。这也融合了单元状态和隐藏状态，还有一些其他的变化。这样得到的模型比标准 LSTM 模型更简单，也越来越为大家所接受，如图 4-44 所示。



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

图 4-44　GRU

以上仅仅是一些最著名的 LSTM 变体。还有许多其他的，像深度门限式 RNN，由 Yao 等人发表。还有一些用完全不同的方法来解决长期依赖性，如时钟型 RNN。

哪些变体是最好的？它们之间有什么差异？Greff 等人做了一个比较流行的变体比较，发现它们都差不多。Jozefowicz 等人测试了超过一万多种 RNN 结构，发现在某些任务中比 LSTM 工作得更好。表 4-1 显示了下一步预测的精度中各个算法的表现情况。

表 4-1　各个算法的表现

Arch.	Arith.	XML	PTB
Tanh	0.29493	0.32050	0.08782
LSTM	0.89228	0.42470	0.08912
LSTM-f	0.29292	0.23356	0.08808
LSTM-i	0.75109	0.41371	0.08662
LSTM-o	0.86747	0.42117	0.08933
LSTM-b	0.90163	0.44434	0.08952
GRU	0.89565	0.45963	0.09069
MUT1	**0.92135**	**0.47483**	0.08968
MUT2	0.89735	**0.47324**	0.09036
MUT3	0.90728	0.46478	**0.09161**

4.7.4 结论

人们用 RNN 取得了显著的成果，这基本上都使用 LSTM 实现，它们对于大多数的任务都工作得很好。

如果作为一个公式呈现出来，则 LSTM 看起来会很吓人。我希望，通过本章对 LSTM 的解析，让 LSTM 显得更通俗易懂。

我们用 RNN 取得了一些成果，而 LSTM 让我们又前进了一大步。RNN 在很多领域还会有其他更多的进步和应用，这就是关注点（attention）。假设你正在使用 RNN 创建一幅图像的标题，意味着它从庞大的图像信息中筛选，并且会选择图像的一部分来对应它输出的每一个词。徐飞飞的论文做得正是这一点，大家感兴趣可以持续研究。

关注点并不是 RNN 研究中唯一令人振奋的地方。比如，Kalchbrenner 等人于 2015 年提出的 Grid LSTMs 似乎非常有前途。Google 在 2016 年提出的 CLSTM，帮助文本预测领域提高了 20%左右的准确度。过去的几年对循环（递归）神经网络来说是令人振奋的时期，而且今后更会如此！

4.8 你是我的眼：利用稀疏编码器找图像的基本单位

本节所用例子引自 https://github.com/danluu/UFLDL-tutorial，运行环境：E3-1230/24GB 内存/256GB SSD，运算速度还是可以接受的。

我们用 MATLAB 实现一个稀疏自编码器，选取吴恩达给的样例，其中数据文件叫作 IMAGES，这是一个 512×512×10 的三维数组，里面存了 10 张图片，每张图片都是 262144 像素。执行：

```
load IMAGES;
imagesc(IMAGES(:,:,10));
```

我们可以看看第 10 张图的样子，如图 4-45 所示，它是一张关于草原和山的

图像。

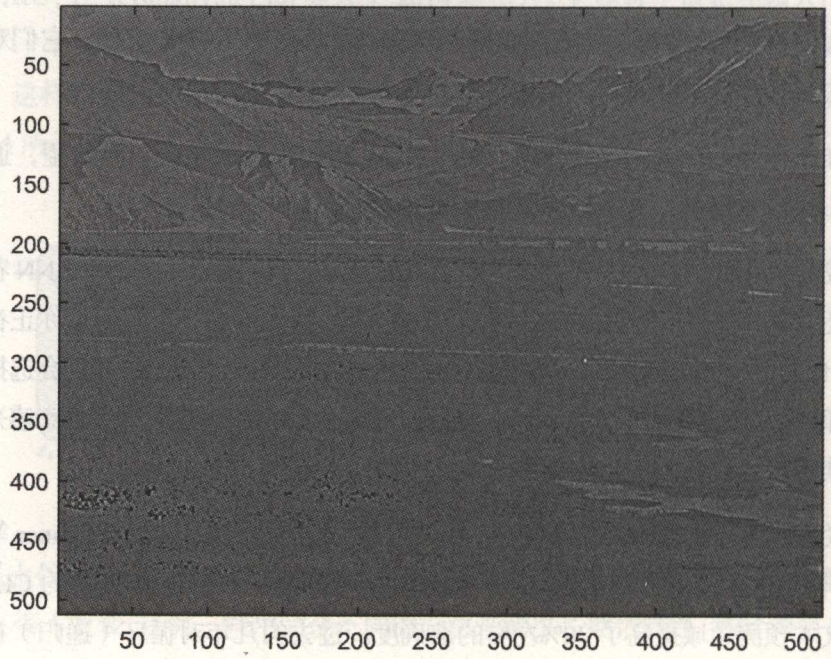


图 4-45　草原和山，取自 UFLDL

数据采样

由于数据的图片挺大，我们也不能一上来就做一个每层都 2 万多节点的神经网络训练。所以先采样，从 10 张图片里随机采样 10000 个小块，每个小块是一个 8 像素 × 8 像素的小碎片，我们定义训练集是 64 × 10000 的矩阵。每一列就是把刚刚的小块拉成列向量的结果。

第一步，定义稀疏自编码器的参数值。

```
visibleSize = 8*8;   % 输入单元数量，8*8 的矩阵一共 64 个单元
hiddenSize = 25;     % 隐层的单元数量
sparsityParam = 0.01;   % 隐层的平均激活值
lambda = 0.0001;     % 权重衰减
beta = 3;            % 稀疏惩罚权重
```

第二步，实现 sampleIMAGES。

调用 sampleIMAGES，display_network 函数可以显示 200 个 patch 的随机样本。

```
patches = sampleIMAGES;
display_network(patches(:,randi(size(patches,2),200,1)),8);
 % 得到随机参数
theta = initializeParameters(hiddenSize, visibleSize);
```

这里的 randi 函数——randi(iMax, m, n)在闭区间[1，iMax]生成 $m \times n$ 型随机矩阵，而 patches 函数的作用是把每个小 patch 拉成一个列向量。

前 200 个 patch 的样子如图 4-46 所示。

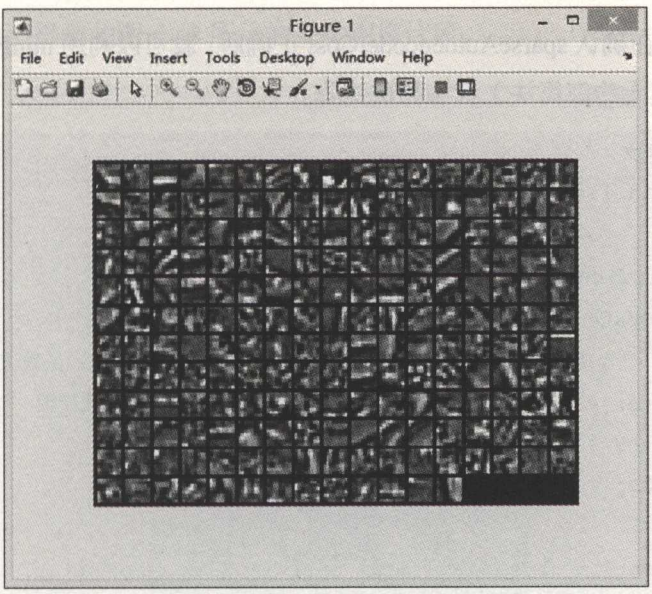


图 4-46 前 200 个 patch

第三步，实现 sparseAutoencoderCost。

```
[cost, grad] = sparseAutoencoderCost(theta, visibleSize, hiddenSize,
lambda,sparsityParam, beta, patches);
```

当均方差项和权重衰减项都调试通过后，得到的图像如图 4-47 所示。

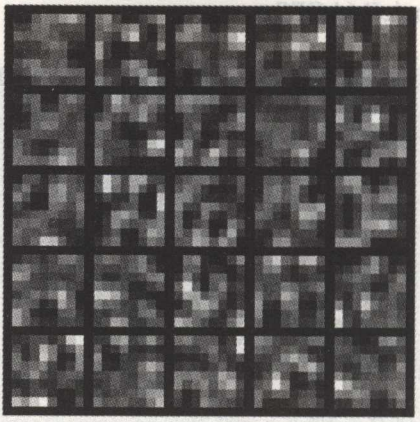


图 4-47　调整参数后得到的图像

第四步，在确认 sparseAutoencoderCost 正确后，就可以利用 minFunc() L-BFGS 算法（大规模无约束算法）训练稀疏矩阵。

```
%  随机初始化参数
theta = initializeParameters(hiddenSize, visibleSize);

%  使用 minFunc 最小化函数，我们使用 L-BFGS 算法优化
options.HessUpate = 'lbfgs'; %这里我们用 L-BFGS 算法来优化
                  %一般来说，为了 minFunc 能工作，需要两个输出：函数值和梯度
options.MaxIter = 400;     % 确定 L-BFGS 运行时最大数量的迭代次数
options.Display = 'iter';
options.GradObj = 'on';

[opttheta, cost] = fminlbfgs( @(p) sparseAutoencoderCost(p, ...
                              visibleSize, hiddenSize, ...
                              lambda, sparsityParam, ...
                              beta, patches), ...
                              theta, options);
```

第五步，可视化，总算可以看看结果了。

```
W1 = reshape(opttheta(1:hiddenSize*visibleSize), hiddenSize, visibleSize);
```

```
display_network(W1', 12);

print -djpeg weights.jpg   % 存储可视化结果
```

目标函数都调试通过后，程序经过 400 次迭代后输出以下结果。

```
Optimizer Results
      Algorithm Used: limited memory BFGS (L-BFGS)
      iterations : 400
      Function Count : 1245
      Minimum found : 0.44988
      Intern Time : 1.2889 seconds
      Total Time : 35.0092 seconds
```

最终，我们得到如图 4-48 所示的基类图像片段。

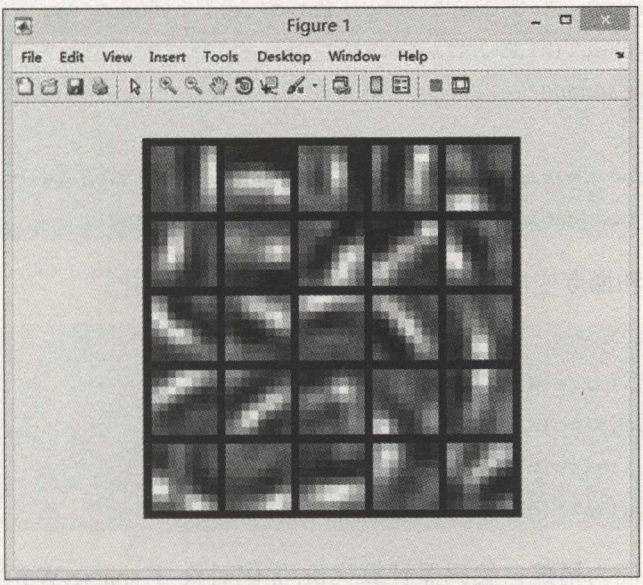


图 4-48 基本图像

这就是图像的基，通过稀疏自编码器，我们学习得到了 25 个 8 像素 × 8 像素的基，这些图像的基就相当于咱们每组视神经细胞所看到的东西，当这些细胞组成阵列，我们就能识别所有的东西了。

解释下 sparseAutoencoderCost 这个方法，这个方法主要计算稀疏自编码器的代价。

visibleSize：输入层单元数量（多数情况是 64）。

hiddenSize：隐层的单元数量（多数情况是 25）。

Lambda：权重衰减参数。

sparsityParam：稀疏参数，隐层单元平均激活量。

beta：稀疏惩罚权重。

data：64 × 10000 的包含训练数据的矩阵。

输入的 theta 是一个向量，我们首先将 theta 转换成(W1, W2, b1, b2)矩阵格式。

```
W1 = reshape(theta(1:hiddenSize*visibleSize), hiddenSize, visibleSize);
W2 = reshape(theta(hiddenSize*visibleSize+1:2*hiddenSize*visibleSize),
visibleSize, hiddenSize);
b1 =
theta(2*hiddenSize*visibleSize+1:2*hiddenSize*visibleSize+hiddenSize);
b2 = theta(2*hiddenSize*visibleSize+hiddenSize+1:end);
```

先把代价和梯度变量（你需要计算出这些值）初始化。

```
cost = 0;
W1grad = zeros(size(W1));
W2grad = zeros(size(W2));
b1grad = zeros(size(b1));
b2grad = zeros(size(b2));
```

接着，我们为稀疏自编码器计算代价/优化对象 J_sparse(W,b)，对应着梯度 W1grad, W2grad, b1grad, b2grad。

我们使用 BP 算法计算这 4 个值，其中 W1grad 和 W1 的大小一样，b1grad 和 b1 的大小相同，依此类推。

W1grad 是 J_sparse(W,b)对 w1 求偏导。

W1grad(i,j)是 J_sparse(W,b)对输入参数 W1(i,j)求偏导。

```
m = size(data, 2);

z_2 = W1 * data + repmat(b1, 1, m);
a_2 = sigmoid(z_2); % 25 10000

rho_hat = sum(a_2, 2) / m;
z_3 = W2 * a_2 + repmat(b2, 1, m);
a_3 = sigmoid(z_3); % 64 10000

diff = a_3 - data;
sparse_penalty = kl(sparsityParam, rho_hat);
J_simple = sum(sum(diff.^2)) / (2*m);
reg = sum(W1(:).^2) + sum(W2(:).^2);

cost = J_simple + beta * sparse_penalty + lambda * reg / 2;
delta_3 = diff .*          (a_3 .* (1-a_3));   % 64 10000
d2_simple = W2' * delta_3;   % 25 10000
d2_pen = kl_delta(sparsityParam, rho_hat);

delta_2 = (d2_simple + beta * repmat(d2_pen,1, m)) .* a_2 .* (1-a_2);

b2grad = sum(delta_3, 2)/m;
b1grad = sum(delta_2, 2)/m;

W2grad = delta_3 * a_2'/m  + lambda * W2; % 25 64
W1grad = delta_2 * data'/m + lambda * W1; % 25 64

%-------------------------------------------------------------------
```

当计算完代价和梯度值后，我们将梯度转换成一个向量格式（配合 minFunc）。

当然，这会展开你的梯度矩阵向量。

```
grad = [W1grad(:) ; W2grad(:) ; b1grad(:) ; b2grad(:)];
```

我们已经将一类图像的基类图像找出来了，接下来大家就可以自由发挥了。第一，你可以做图像识别和分类检测；第二，你可以随意组合这些图像基类；第三，你可以将图像识别后结合自然语义分析转换成一段语言描述它（这个过程是完全自动化的）。

4.9 你是我的眼（续）

刚才，我们用稀疏自编码器找到图像的基，然后利用这些图像的基去做图像识别等一系列动作，那有没有更为简单方便的算法，让我们点一点鼠标就能产生一个算法从而识别图像呢？

回答是：当然有！还记得神经网络中我们用到的 Neuroph 吗？其实 Neuroph 不仅仅是个 Java 的神经网络库，它还有个 Neuroph Studio。这个 Studio 是什么东西呢？

Neuroph Studio 是 Neuroph 中的一个可以快速搭建和训练神经网络的工具集，它提供了类似图形化的界面帮你完成神经网络的搭建，它甚至还有一个专门的图像识别工具去训练神经网络识别图像。

我们将按照以下步骤用 Neuroph Studio 建立一个神经网络并训练它识别图像，做完整个过程不会超过 1 小时！

（1）建立一个 Neuroph 项目。

（2）建立一个图像识别神经网络。

（3）训练这个网络。

（4）用图像测试相关数据。

（5）保存并分发这个神经网络。

第一步，建立一个神经网络，选择 File → New Project 选项，如图 4-49 所示。

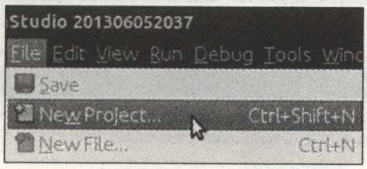


图 4-49 新建项目 1

选择 Neuroph Project 选项并单击“Next”按钮，如图 4-50 所示。

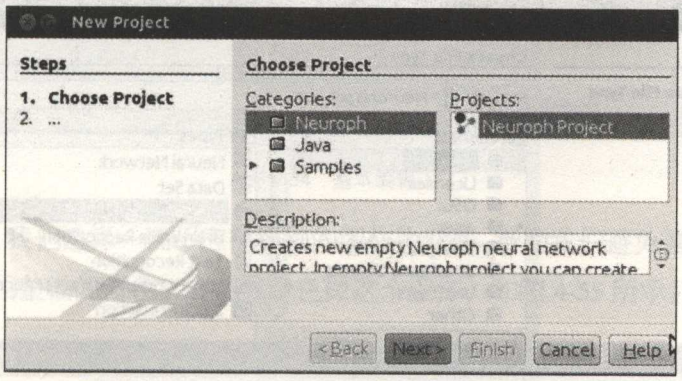


图 4-50 新建项目 2

输入项目名称和路径，单击“Finish”按钮，如图 4-51 所示。

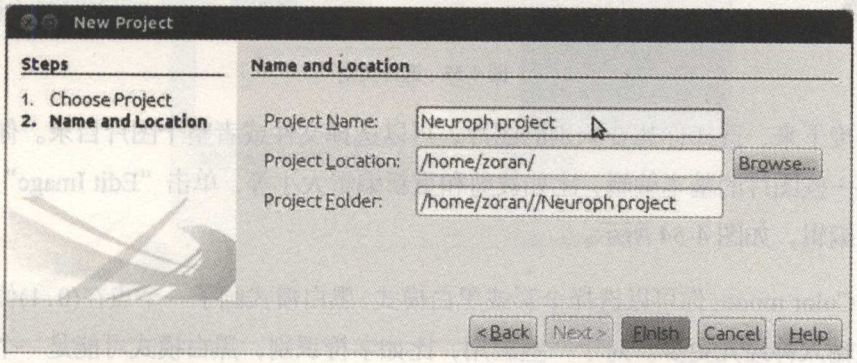


图 4-51 新建项目 3

第二步，建立一个图像识别网络，选择 File→New File 选项，如图 4-52 所示。

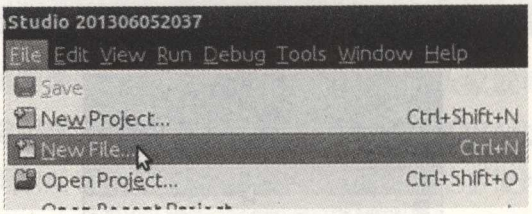


图 4-52　新建文件

选择 Image Recognition 选项，并单击“Next”按钮，如图 4-53 所示。

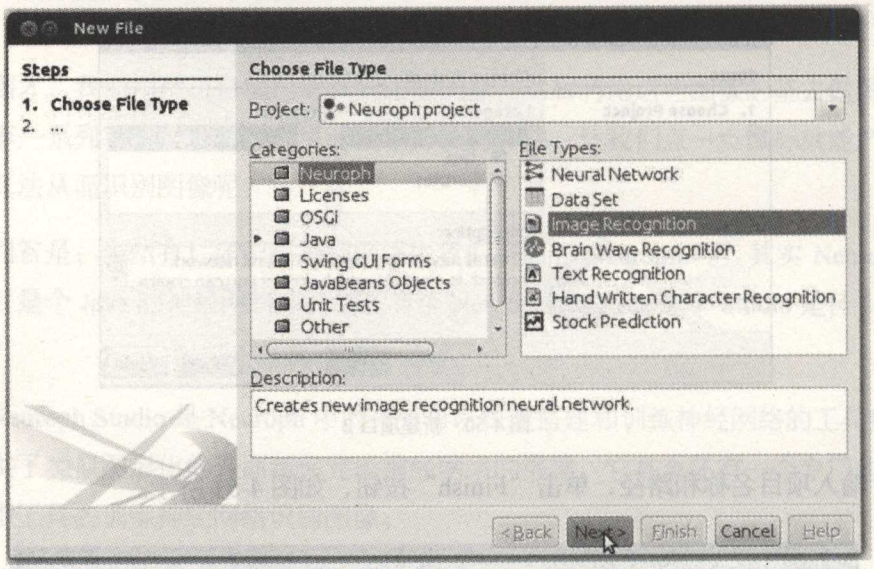


图 4-53　选择类型

接下来，选择你想要识别的图片，可以选择文件或者整个图片目录。你也可以做一些图片的基本编辑，比如裁剪和重新编辑大小等，单击“Edit Image”按钮进行编辑，如图 4-54 所示。

Color mode：你可以选择全彩或黑白模式。黑白模式画了一个点在(0，1)位置，并且输入神经元更少。对于一些应用，比如字符识别，黑白模式可能是一个更好的选择。

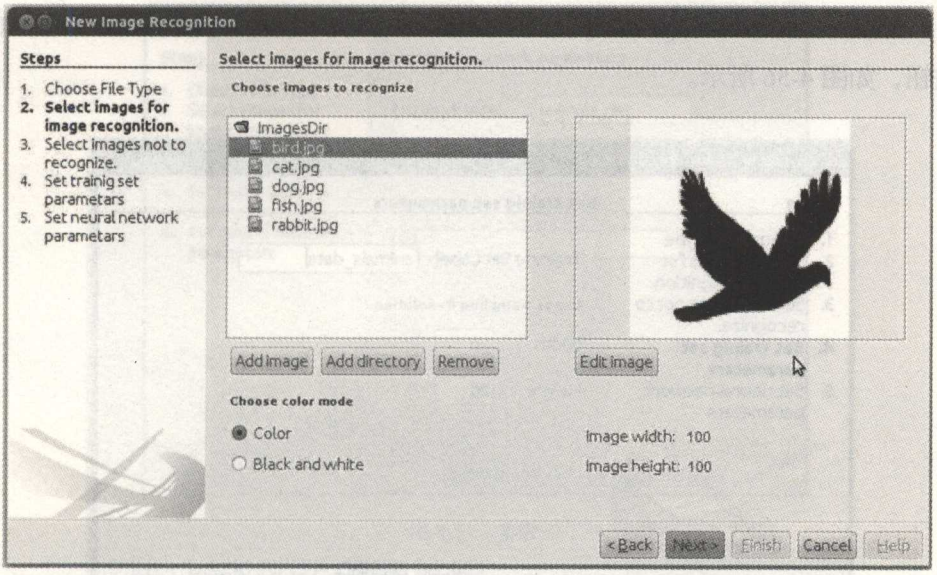


图 4-54 基本编辑 1

接下来，我们选择一幅不应被识别的图作为背景，帮助我们避免错误的识别。通常，我们用整幅的蓝色、红色或绿色做这个工作，如图 4-55 所示。

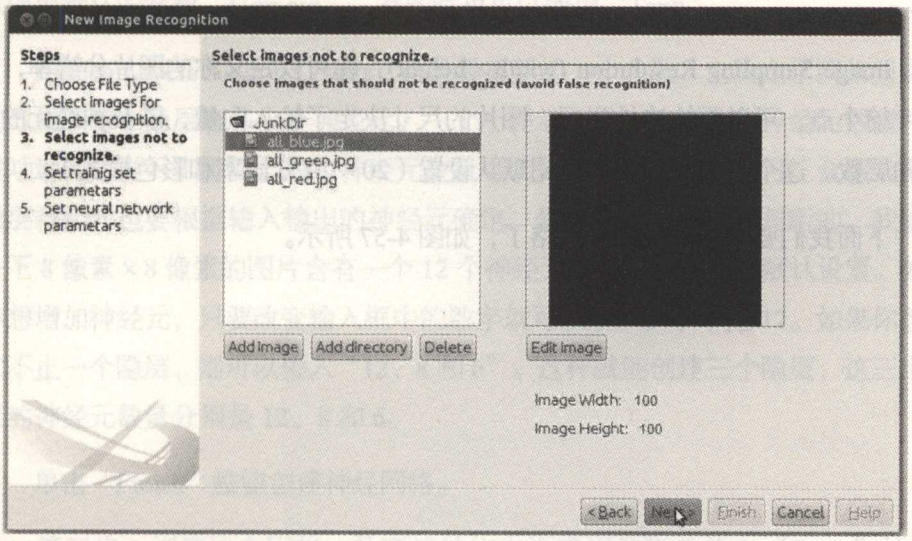


图 4-55 基本编辑 2

然后，进入 Training Set 标签页做图像样例分辨率设置，完成后单击“Next”按钮，如图 4-56 所示。

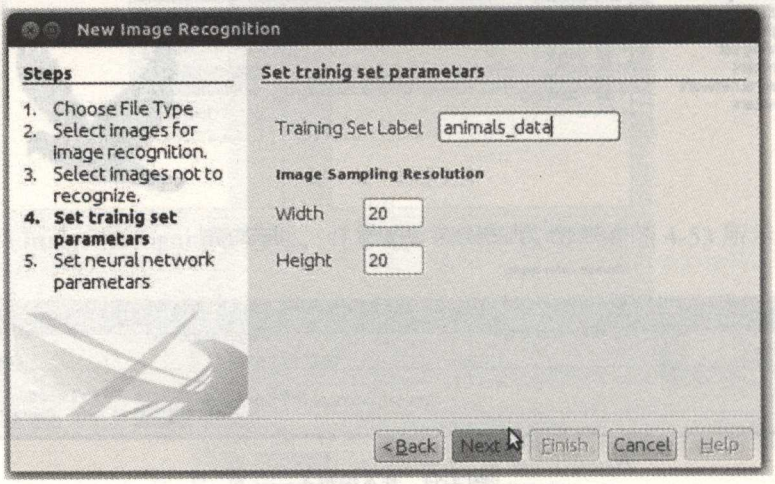


图 4-56　训练参数 1

Training Set 标签页：尝试网络的时候可以设置多个训练集，对标记有很大帮助。

Image Sampling Resolution (width × height)：你可以定义你的图片分辨率，把图片缩小点，可以更快速地学习。图片的尺寸决定了输入向量，以及神经元的个数和层数。这个案例中，我们使用默认设置（20 × 20 分辨率和彩色模式）。

下面我们要创建一个神经网络了，如图 4-57 所示。

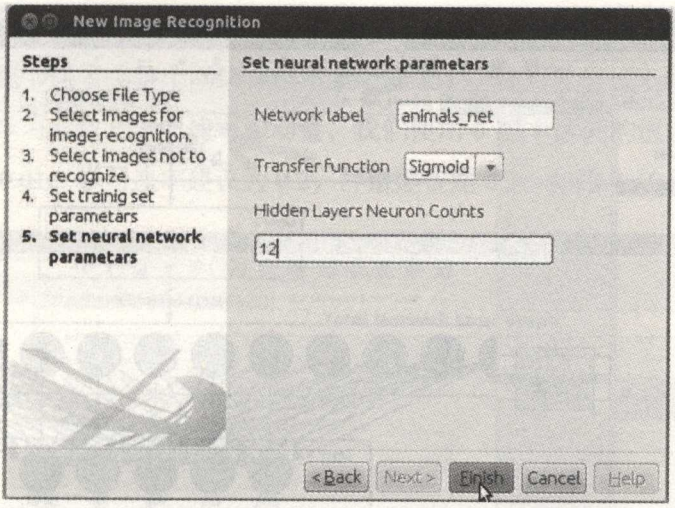


图 4-57 参数

创建一个神经网络，你可以按照下面的步骤完成。

Network Label：神经网络的名称，发生错误时，可以按此比对。

Transfer Function：这个设置决定了神经元的传输函数。大多数情况下，你可以保留默认的参数“Sigmoid”，有些时也可以使用“Tanh”。

Hidden Layers Neuron Counts：这是最重要的一个参数，它决定了网络里隐层的数量和每个隐层有多少个神经元。一个小提示：最小的层数和神经元数量可以成功地完成学习训练集，较少神经元数量可以更快更好地完成学习。合适数量的隐层神经元也要根据输入输出的神经元确定，最好的值是凭经验判断的。我们试一下 8 像素×8 像素的图片含有一个 12 个神经元的隐层，这也是默认设置。如果你想增加神经元，只要改变输入框中的数字就可以了，例子中是 12。如果你想增加不止一个隐层，则可以输入“12、8 和 6”。这样就能创建三个隐层，这三个隐层的神经元数量分别是 12、8 和 6。

单击“Finish”按钮创建神经网络。

第三步，训练这个网络。从项目结构上选择训练集并单击“Train”按钮，如图 4-58 所示。

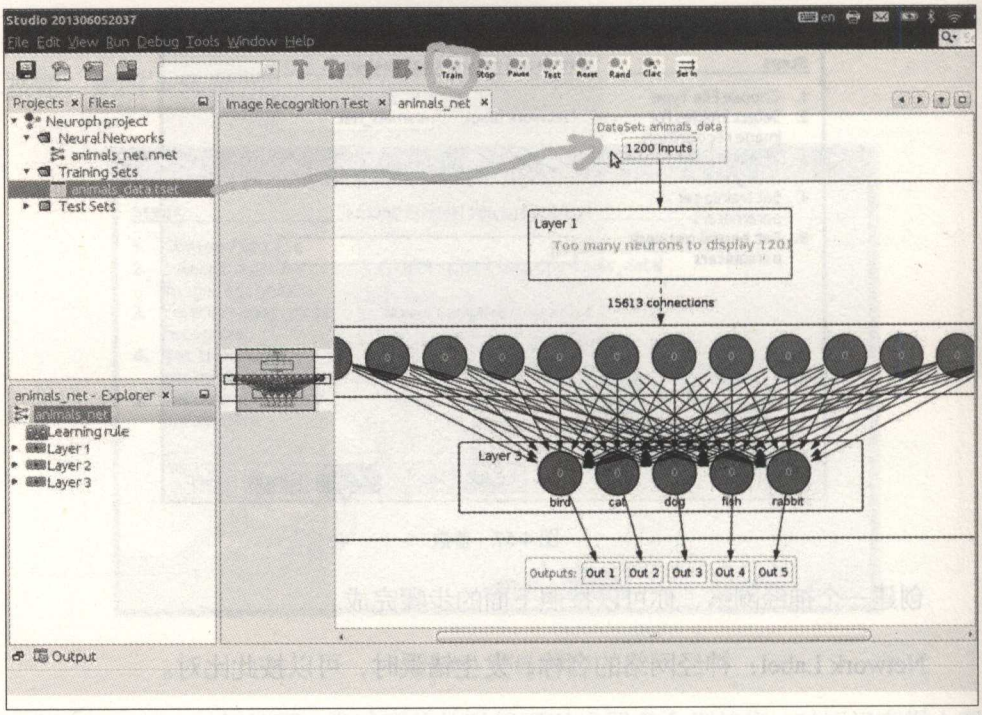


图 4-58　训练

此时会打开一个对话框，设置学习参数。我们使用默认的学习设置，单击“Train”按钮，如图 4-59 所示。

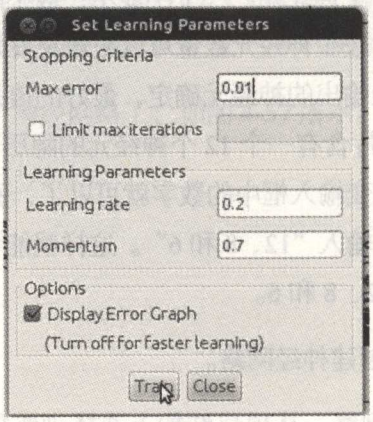


图 4-59　训练参数 2

然后，工具会自动打开网络学习图像和迭代计数器，便于你观察学习的进程。如果这个过程卡住了（整个网络错误不会下降了），你可以尝试不同数量的神经元、层数和学习参数、学习速率和动量，我们使用 0 到 1 区间的值[0,1]，学习错误的值推荐 0.01。学习速率建议为 0.2，冲量建议是 0.7，如图 4-60 所示。

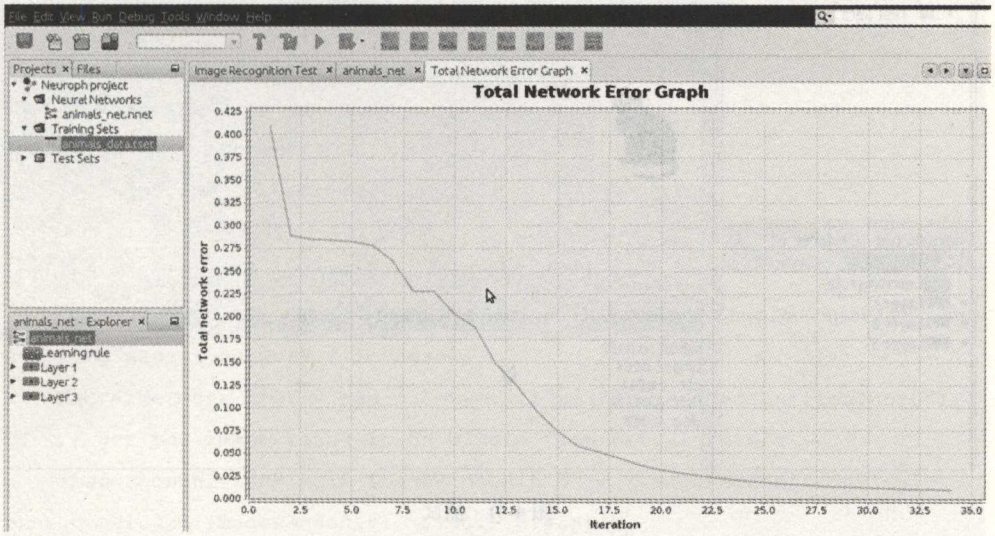


图 4-60 训练过程

第四步，测试神经网络。

在训练完网络后，你可以尝试让它跑跑看。单击“Select Test Image”按钮为网络设置输入图片，网络将输出图片标签列表和对应的值，识别出来的图片对应了最高的数值。单击“Test whole data set”按钮，可以测试整个数据集，如图 4-61 所示。

第五步，存储神经网络。

以 Java component 的形式存储神经网络，单击 File→Save 菜单命令，并使用.nnet 扩展名，这个网络将被存储为 MultiLayerPerception 对象，如图 4-62 所示。

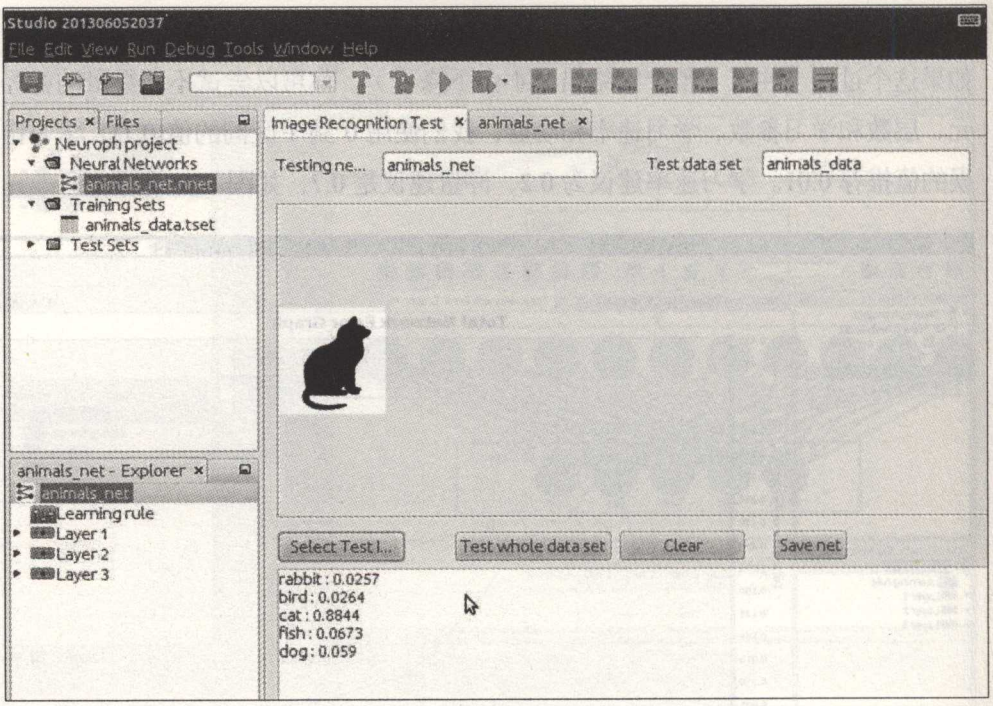


图 4-61　测试

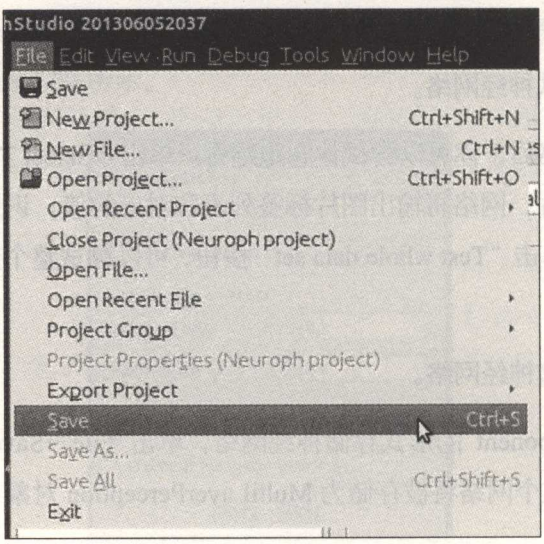


图 4-62　存储神经网络

在你的程序中使用 Neuroph Image Recognition。

以下样例代码显示如何使用经过训练的图像识别神经网络。你可以试试这个样例，注意根据需要更改文件名和测试图像。

```
import org.neuroph.core.NeuralNetwork;
import org.neuroph.contrib.imgrec.ImageRecognitionPlugin;
import java.util.HashMap;
import java.io.File;
import java.io.IOException;
public class ImageRecognitionSample {

 public static void main(String[] args) {
    // load trained neural network saved with Neuroph Studio (specify some
existing neural network file here)
    NeuralNetwork nnet = NeuralNetwork.load("MyImageRecognition.nnet");
    // get the image recognition plugin from neural network
    ImageRecognitionPlugin imageRecognition = (ImageRecognitionPlugin)
nnet.getPlugin(ImageRecognitionPlugin.class);
    try {
         // image recognition is done here (specify some existing image file)
        HashMap<String, Double> output =
imageRecognition.recognizeImage(new File("someImage.jpg"));
        System.out.println(output.toString());
    } catch(IOException ioe) {
        ioe.printStackTrace();
    }
 }
}
```

图像识别也可以用另外一种方法从 ImageRecognitionPlugin 调出：

```
imageRecognition.recognizeImage(new File("someImage.jpg"));
```

ImageRecognitionPlugin 提供神经网络的图像识别接口。你甚至可以直接识别

互联网上的图片，比如：

```
imageRecognition.recognizeImage(new URL("http://www.example.com/
someImage.jpg"));
```

可能出现如下问题。

（1）将图片缩放成一样尺寸可以避免一些问题。

（2）使用同样色彩和尺寸的图片用于识别。一般来说，黑白图片训练得更快。

（3）如果报内存异常则可以调整 Java 虚拟机参数，比如 JVM –Xms –Xmx。

4.10 使用深度信念网搞定花分类

我们来看看 DL4J 给出的例子，链接为 http://deeplearning4j.org/iris-flower-dataset-tutorial.html。

DBN 算法我们之前讲过，它是多个 RBM 层叠起来的多级分类器模型。鉴于许多输入属于不同类别，DBN 可以先从一个小训练集开始学习，根据这些不同类别，不需要分类就可以根据数据记录，决定一组输入数据的标签。

把 DBN 训练好后，给它一个输入，DBN 会从一组标签选择并返回一个适合的标签。这比简单的是或否的标签分类选择更多。

也就是说，网络输出向量的每个输出神经元节点包含一个数字（这和我们之前讲到的神经网络的例子非常像），而输出神经元节点的数量就等于标签数。每个输出都会是一个 0 或 1，最终，我们根据输出的 0 和 1 加在一起构成的矢量，判断出花的类别。

1. 鸢尾花数据集（IRIS Dataset）

鸢尾花数据集广泛应用于机器学习来测试分类技术。我们将用这个数据集验证我们的网络有效性。

先从这个数据集中取出3种鸢尾花共50个样本，总共有150朵花和600个数据。3种花的种类分别是Iris-setosa、Iris-Virginica和Iris-Versicolor。将花的萼片、花瓣的长度和宽度分别定义，共有4个数据（Sepal length、Sepal width、Petal length和Petal width。最后，我们将这4类描述鸢尾花的特性的数据作为输入数据集（其中花的种类作为训练的标签数据集）。

接着，我们在深度信念网上用这些数据集创建一个完美的测试。只用这4个特征足以准确地分类3种鸢尾花。换句话说，在只知道特征尺寸的前提下，我们就可以用一个神经网络成功地进行鸢尾花分类。如果分类失败，则需要修正神经网络参数，直到成功为止。

花的样本分类如表4-2所示。

表4-2 花的样本分类

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	i.setosa

DL4J的算法需要如下形式的数据输入：

```
5.1,3.5,1.4,0.2,i.setosa
```

接着，我们将这些单词都去掉，把这些数值重新赋予两个对象：

```
Data: 5.1,3.5,1.4,0.2
Label: 0,1,0
```

三个输出节点转换成二进制数字，将这三种鸢尾花标记为三种类型，我们用[1,0,0]、[0,1,0]或[0,0,1]表示。

用DL4J加载这些数据。DL4J使用DataSetIterator和DataSet对象将数据加载到一个神经网络。一个DataSet拥有数据和它关联的标签。这个DataSetIterator负责从原始文件中读取这些数据，如表4-3所示。

表 4-3 叫声和动物的对应关系

First(data to predict)	Second(outcome,or labels)
ribbit	frog
bark	dog
meow	cat

DataSet 对象内有两个 NDArrays，这里 NDArrays 适用于复杂的数学运算，并经常用于科学计算。

如果读者是程序员，则应该熟悉 CSV（逗号分隔型）文件中的数据集，鸢尾花数据集也不例外。这里会演示如何分析包含鸢尾花数据的 CSV ，并把它变成一个 DL4J 可以理解的对象。

```
File f = new File("Iris.dat");
InputStream fis = new FileInputStream(f);
List<String> lines = org.apache.commons.io.IOUtils.readLines(fis);
INDArray data = Nd4j.ones(to, 4);
List<String> outcomeTypes = new ArrayList<>();
double[][] outcomes = new double[lines.size()][3];
```

让我们具体解释：iris.dat 是一个 CSV 文件，里边含有我们需要输入神经网络的数据。

这里面，我们使用 IOUtils，一个 Apache 库，从文件中读取数据流中的数据。readLines 方法会将所有数据复制到内存（一般生产环境中不要这样做），在生产环境中可以使用一个 BufferedReader 对象。

该 NDArray 会保留我们的原始数字，而 List<String> outcomeTypes 是一个包含标签的列表。该数据集 DataSet completedData （在以下代码的最后一行）包含了所有的数据，也包括了给定的标签。

我们会看到一个二维数组，这个二维数组具有我们所有记录的行（即在 iris.dat 的行数据），也具有我们所有的标签（即花的 3 个种类）。

请看下面这段代码：

```
for(int i = from; i < to; i++) {
   String line = lines.get(i);
   String[] split = line.split(",");

    // turn the 4 numeric values into doubles and add them
   double[] vector = new double[4];
   for(int i = 0; i < 4; i++)
       vector[i] = Double.parseDouble(line[i]);

   data.putRow(row,Nd4j.create(vector));

   String outcome = split[split.length - 1];
   if(!outcomeTypes.contains(outcome))
      outcomeTypes.add(outcome);

   double[] rowOutcome = new double[3];
   rowOutcome[outcomeTypes.indexOf(outcome)] = 1;
   outcomes[i] = rowOutcome;
}

DataSet completedData = new DataSet(data, Nd4j.create(outcomes));
```

第 3 行：处理 CSV 数据，我们可以使用 split 按逗号分隔，拆分这些数据然后把它存在 String 数组里面。

第 6～10 行：向 String 对象里存放数字。我们将创建一个临时数组 vector，保存以备后用。

第 12～14 行：通过获取 String 数组的最后一个元素来取得标签。比如二进制化的标签。为了做到这一点，我们将所有的标签列表中的 outcomeTypes 收集起来。

第 16～18 行：开始使用 outcomeTypes 列表二进制化这些标签。每个标签都

有一定的状态或索引，我们将该索引号映射到创建的标签行。如果 i.setosa 是标签，我们把它放在 outcomeTypes 列表的末尾。然后创建一个新的标签行，大小为三个元素，并在 rowOutcome 的对应位置标为（1,0）。最后，我们将 rowOutcome 保存到我们之前创建的二维数组。

完成后，将得到一排带着数字表示的标签，如表 4-3 所示。

表 4–3　用数字表示花的各部分特征

Sepal length	Sepal width	Petal length	Petal width	i.setosa	i.virginica	i.versicolor
5.1	3.5	1.4	0.2	1	0	0

第 1 行的词只是为了比较清晰地标注数字的属性，也顺便对应属性和数字便于查看。表的下部的数字会以向量的形式来处理数据。底部就是我们所说的量化数据。

第 21 行：现在可以开始包装 DL4J 的数据。为了做到这一点，我们创建包含二进制标签的单个 DataSet 对象。

最后，返回 completedData 列表，这是一个能在深度信念网运行的数据集。

2. 创建一个神经网络

现在，我们准备建立一个深度信念网络或 DBN 来做分类。

首先，需要创建一个神经网络的配置对象：

```
NeuralNetConfiguration conf = new NeuralNetConfiguration.Builder()
.hiddenUnit(RBM.HiddenUnit.RECTIFIED).momentum(5e-1f)
    .visibleUnit(RBM.VisibleUnit.GAUSSIAN).regularization(true)
    .regularizationCoefficient(2e-4f).dist(Distributions.uniform(gen))
    .activationFunction(Activations.tanh()).iterations(10000)
    .weightInit(WeightInit.DISTRIBUTION)
.lossFunction(LossFunctions.LossFunction.RECONSTRUCTION_CROSSENTROPY).rn
g(gen)
    .learningRate(1e-3f).nIn(4).nOut(3).build();
```

这个对象拥有 DBN 分类器所需要的所有东西。这里有很多的参数，你可以慢慢学会如何调整神经网络以提高它的性能和准确度。

这些参数包括动量、布尔型（是或否）及其系数、迭代的次数、学习速率、输出节点的数量，以及连接到每个节点层的传输函数（如高斯 Gaussian 或整流）。

我们还需要一个随机数产生器对象：

```
RandomGenerator gen = new MersenneTwister(123);
```

最后，创建 DBN 本身：

```
DBN dbn = new DBN.Builder().configure(conf)
    .hiddenLayerSizes(new int[]{3})
    .build();
  dbn.getOutputLayer().conf().setActivationFunction(Activations.
softMaxRows());
  dbn.getOutputLayer().conf().setLossFunction(LossFunctions.
LossFunction.MCXENT);
```

现在来分析上面的代码：在第一行，把 conf 的配置对象作为一个参数传入。然后指定隐层的大小，我们可以控制每个层的节点数量。在这个例子里，有一个隐层包含三个神经元。

准备好刚才做的 DataSet 对象并把它放在一个单独的 loadIris()函数中。

```
DataSet ourDataSet = loadIris(0, 150);
ourDataSet.normalizeZeroMeanZeroUnitVariance();
dbn.fit(ourDataSet);
```

注意上面的第二行。在很多机器学习模型里，标准化数据表示是非常重要的，需要将数据都规则化后放入[0，1]区间，这种数据规则化也叫归一化。

最后，调用训练模型的数据集。通过对数据集的训练，算法会自动学习提取数据的一些特定特征，这些特征帮助我们根据输入数据将花区分成各个种类。

训练是基于机器提取的特征并尝试分类。这里有两个主要步骤：比较输出值与测试组中的原数据；当它接近或远离正确的答案时会有“奖励”或“惩罚”。有了足够的数据和训练之后，神经网络会分类无标签的鸢尾花数据，而且精度将会相当高。

如果开启 Debug 模式，则你会在运行结果的最后观察到一些输出。

评估我们的结果，在使用 fit()后，考虑使用以下代码。

```
Evaluation eval = new Evaluation();
INDArray output = d.output(next.getFeatureMatrix());
eval.eval(next.getLabels(),output);
System.out.printf("Score: %s\n", eval.stats());
log.info("Score " + eval.stats());
```

DL4J 有个叫评估对象（Evaluation Object）的类别，可以通过调用它来收集有关模型性能的统计信息。该 INDArray output 是由 DataSet.getFeatureMatrix()创建的。

getFeatureMatrix()将返回所有数据输入，eval 负责收集模型的结果和实际结果。

评估对象包含了许多有用的调用，如 F1()。这个方法会使用概率的形式来估计模型的准确度（下面这个结果表示模型认为分类能力有 77%的准确率）。其他方法包括 precision()：它告诉我们模型预测结果的可靠性（相同的输入下）；而 recall()会告诉我们有多少个正确结果。

在这个例子中，我们得到以下结果：

```
Actual Class 0 was predicted with Predicted 0 with count 50 times

Actual Class 1 was predicted with Predicted 1 with count 1 times

Actual Class 1 was predicted with Predicted 2 with count 49 times

Actual Class 2 was predicted with Predicted 2 with count 50 times

====================F1Scores========================
```

```
0.767064393939394
==================================================
```

网络经过训练后，你会看到这样的 F1 分数。在机器学习中，F1 是一个度量分类器好坏的指标。它是一个[0,1]区间的数字，会告诉你这个神经网络的准确率，1 相当于 100%的预测准确率。

我们的模型没有好好地调整参数，但作为第一步，这已经比较不错了。

最后，图 4-63 所示为鸢尾花数据集的可视化图形。

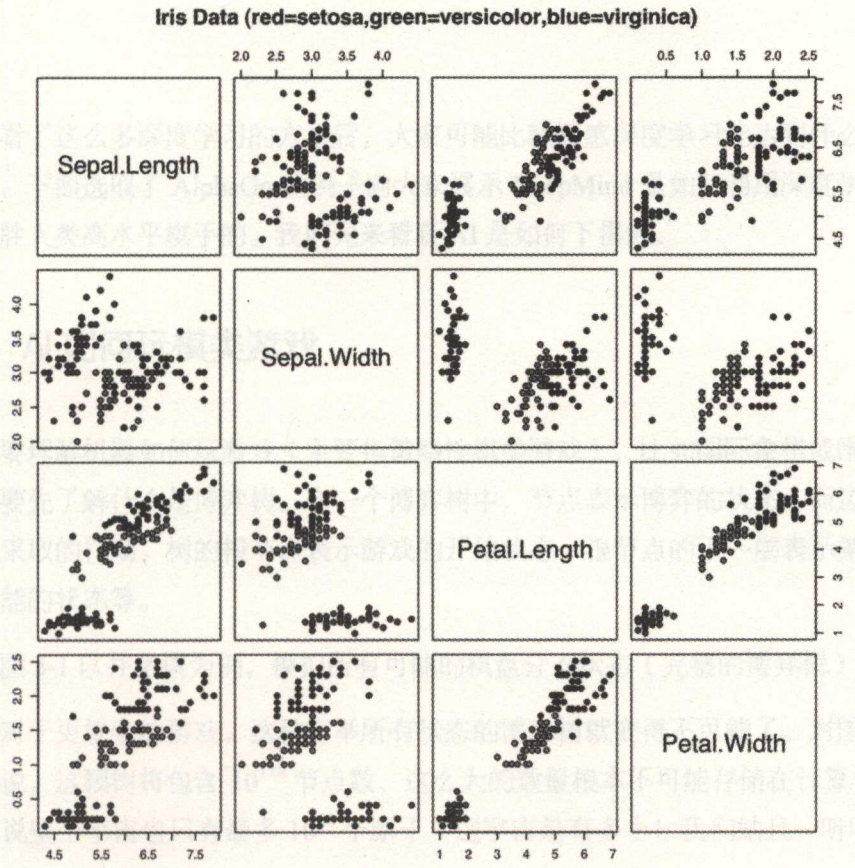


图 4-63 鸢尾花数据集可视化图形

第 4 章部分图片引自如下网址。

[1] A Deep Learning Tutorial: From Perceptrons to Deep Networks
http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks

[2] An Introduction to Convolutional Neural Networks
http://white.stanford.edu/teach/index.php/An_Introduction_to_Convolutional_Neural_Networks

5 深度学习的胜利：AlphaGo

看了这么多深度学习的方法后，大家可能比较疑惑深度学习能达到什么样的水平，下面选取了 AlphaGo 的例子向大家展示 DeepMind 是如何利用深度学习技术战胜人类高水平棋手的。我们先来看看 AI 是如何下棋的。

5.1 AI 如何玩棋类游戏

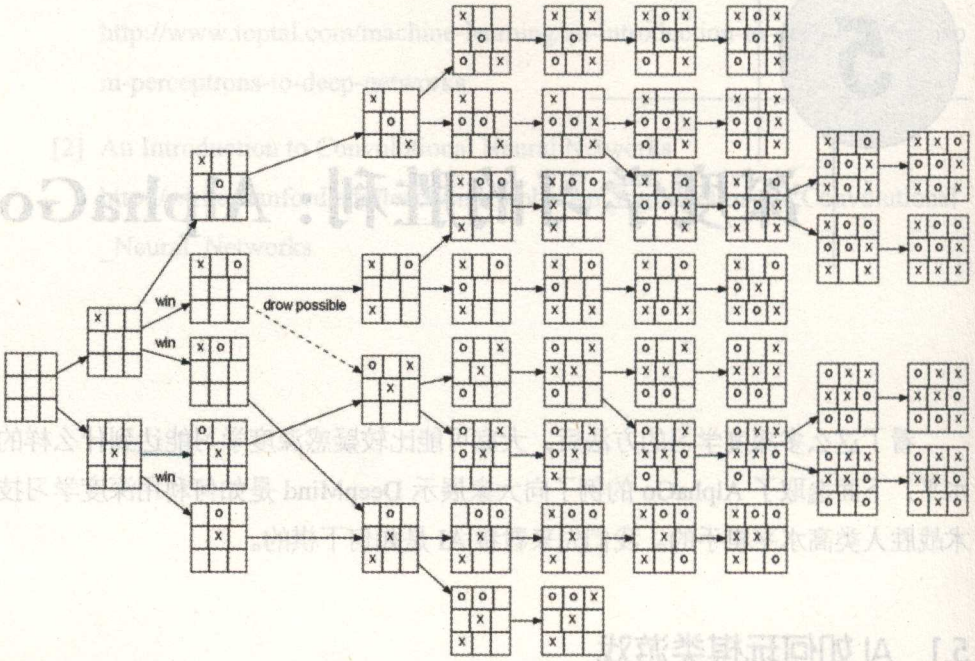
要理解机器如何玩游戏（主要指策略性棋类游戏），比如国际象棋或围棋，我们要先了解什么是博弈树。在一个博弈树中，节点表示博弈的状态，而边表示可能采取的行动。树的根节点表示游戏的开始状态。根节点的下一层表示第一步后可能的状态等。

图 5-1 以井字棋为例，模拟所有可能的棋盘分支状态（完整的博弈树）。

对于更复杂的游戏，这种穷举所有状态的博弈树就变得不可能了。对国际象棋来说，这颗树将包含 10^{120} 节点数，这么大的数量根本不可能存储在计算机上，有人说整个宇宙也只有最多 10^{80} 个原子（这宇宙是有多小！我们姑且一听吧）。

对于一个玩游戏的 AI 来说，知道整个博弈树是非常有用的，因为它允许程序在某个游戏状态下挑选一个最好的移动。我们可以用最大最小化算法（Minimax

Algorithm）来计算：即在每次游戏的回合中，AI 会计算出，将最坏的情况减小到最低应该怎么做。



（资料来源：维基百科）

图 5-1　井字棋的完整博弈树

怎么做到这一点呢？首先，找到树中相对的节点，反映游戏的当前状态，然后把可能发生的最坏情况减少到最低。这需要在整个博弈树中，将所有可能的路径搜索一遍，一直搜索到游戏结束的状态，也就是树的底层节点。

所以，最大最小化这种算法需要完整的博弈树，而且从中可以看到，这对于比较简单的井字棋帮助很大，因为可以穷举整个博弈树，但对国际象棋根本没有用，更不用提围棋了。

那么深蓝系统是如何击败卡斯帕罗夫的呢？采用的方法是，深蓝的博弈树去搜索后面的六步，我们用树来表示是搜索到第六层。然后，利用一个评估函数（Evaluation Function）在这个层次上评估节点的质量。从本质上讲，评估函数用一个值替换了这个节点之下的所有子树的概述，也就是不计算每种移动，而是给

出当前移动后胜利的概率。然后，深蓝运行了一个同样的极大极小算法：按照将最坏的情况减到最低的原则来选择下一步动作。

5.2 围棋的复杂性

在国际象棋中，每个棋子的价值是不一样的。马或者象价值三个兵。车的活动范围更大，相当于五个兵。王后是所有棋子中可移动范围最大的，相当于九个兵。国王的价值最高，因为失去它就输掉了比赛。

棋局过程中我们基本上是根据这样的价值来估算下一步是不是值得。用象来换你对手的车？这是一个好想法。用马来换你对手的车？这就不是什么好主意了。

价值观念在下国际象棋时可以起非常重要的作用。大多数国际象棋程序都是通过数亿次的模拟移动和对弈来决定走法。目标是无论对手怎样走棋，程序都要找到一个能最大化整体价值的走法——也就是赢得比赛胜利的走法。

早期的象棋程序使用简单的概念评估棋盘上的局势，类似于我们刚才讲的“一个象等于三个兵”。后来，程序中使用了更详细的象棋知识。比如深蓝，用了超过 8000 个不同的因素评估棋盘局势。深蓝不只会认为一个车相当于五个兵。如果相同颜色的兵挡在车之前，会限制车的运动范围，从而降低车的价值。但是在这种情况下，如果把车移动出来，去捕获对方的兵，深蓝就会认为车的价值减少了。

深蓝的程序显然要依赖于对国际象棋知识的深入理解，这是深蓝战胜人类至关重要的一点。据深蓝团队撰写的技术论文所述，像上一段所讲那样，深蓝对车和兵价值的理解，是深蓝对抗卡斯帕罗夫第二场比赛的关键。

最终，深蓝的开发人员使用了两种主要想法。第一种是建立一个纳入许多详细的象棋知识，可以评估任何棋盘局势的功能。第二种是利用巨大的计算能力来评估大量可能的局势，挑选出下一步的走法，使当前局势趋向一种最有利的结果。

现在，让我们想一想，如果用深蓝的策略去下围棋，会发生什么？

事实上，如果这样的话，会遇到一个棘手的问题。问题的关键在于如何判断棋盘上的局势。很多顶尖棋手会用直觉判断一盘棋目前的局势。这种判断即使在AlphaGo对弈李世石时，解说员也在频繁使用。人类目前也不清楚该怎样表达这种对棋局判断的直觉上定义良好的棋局评估系统。

小知识：决定AI实力的因素

两个因素决定了AI的实力。

纯计算能力：更强大的计算能力意味着博弈树能搜索到一个更大深度，也就能使评估函数得到一个更好的估计值。深蓝当时运行在一台超级计算机上，或许那是当时顶级的计算机了，这也表示它有强大的计算能力。

评估函数的质量：IBM把大量的精力投入到评价函数的设计上，按照维基百科的说法：评估函数被分为8000个部分，其中很多是为特殊的位置设计的。这个数据库里有超过4000个位置和超过700000局国际象棋游戏。终局数据库中包含6种终局和5个或更少的形式。在第二局开始前，程序的国际象棋专业知识被象棋大师Joel Benjamin很好地调整过。这个开放库提供了象棋大师Miguel Illescas、John Fedorowicz和Nick de Firmian的知识。

总之，尽管国际象棋非常复杂，深蓝依靠蛮力，外加一些启发式算法设计搞定了它。

围棋不能用像国际象棋那样的方法解决。围棋比国际象棋每一步都有更多可能的落子点：围棋是200种，国际象棋是37种。围棋的对弈时间往往更长：对于围棋来说，直到决出胜负，双方都可能有200次落子，对于国际象棋来说只有57次。围棋可能的落子位置有10的170次方，而国际象棋只是10的47次方。围棋有10的360次方这么多种合规的落子组合序列，国际象棋的这个数字是10的123次方。这使得博弈树搜索到足够深度更困难。此外，围棋评估函数的设计比国际象棋更困难，围棋的终局有时候特别复杂，2016年3月15日维基百科中相关内

容更新为：不太可能有一种快速的算法使终局完美，更不要说整盘棋局。从AlphaGo最近赢得的比赛来看，这个说法是比较悲观的，当然也是错的。

AlphaGo是如何用事实证明维基判断失误了呢？我们先来看看原理。

5.3 AlphaGo的主要原理

如果需要了解全部细节，请读者通读其他相关论文。本节只为大家展示最重要的组件并进行说明。AlphaGo是由三个不同部分组合成的。

（1）估值网络（Value Network也叫价值网络）。它估计棋局的状态（运行时没有进行任何搜索动作），计算谁领先了，并且领先了多少步——从技术上讲，它估计每一方赢的概率，同时假设每一方都是由AlphaGo扮演的（AlphaGo不能为对手建模，所以它总是和自己下棋）。

（2）走棋策略网络（Policy Network）。它决定了棋局的状态并且选择下一步的走法，运行时也没有进行任何搜索。它首先由专家训练，并且预测他们下一步怎么走。然后，它跟自己下棋，下数百万次后，再训练指导系统的下一步，直到最终获得胜利，如图5-2所示。

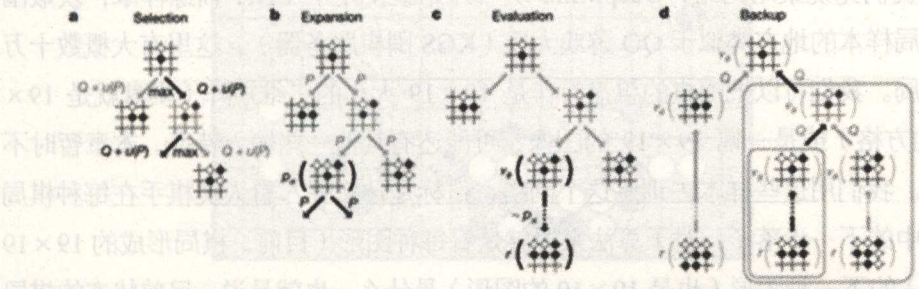


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b,** The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c,** At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d,** Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

图5-2 走棋策略的网络

（3）树搜索（MCTS，Monte Carlo Tree Search，蒙特卡洛树搜索）。最后，树搜索把两个网络结合在一起，模拟下一步会发生什么，并通过策略网络选择最佳的落子位置。AlphaGo 的原理可以简单总结为：首先通过估值网络评估棋局情况，其次通过一个快速的策略网络选择下一步的位置，一直下到最后，胜利！

5.3.1 策略网络

我们简单地了解了 AlphaGo 的技术背景，再看看 DeepMind 是如何打造 AlphaGo 的。

首先，AlphaGo 有两个大脑，一个是走棋策略网络，一个是估值网络。无论是策略网络还是估值网络都是一个 CNN 结构。这个 CNN 有 13 层，卷积核大小为 5×5。为什么要用 CNN 去做计算呢？棋盘是一个 19×19 的方格，每个方格像一个像素点，整个棋盘就像 19×19 的图片一样。如果把它当作图片来处理，我们第一个想到的就是能用深度学习压缩表示，第二个想到的是能做基本分类。

如果我们已经确定了两个网络的结构，那么深度学习算法还需要解决训练数据获取的问题。

我们先说策略网络，DeepMind 从一开始就使用了 CNN 训练样本，获取围棋对局样本的地方类似于 QQ 游戏大厅（KGS 围棋服务器），这里有大概数十万个棋局。我们可以把围棋的棋盘看作是 19×19 大小的二维矩阵（棋盘就是 19×19 的方格）或是一幅 19×19 的图像，可能还有其他一些输入特征，本章暂时不探讨。我们用这些样本去训练这个网络，目标是什么呢？看人类棋手在每种棋局状态中的下一步落子，对于算法来说就是看每种图形（目前，棋局形成的 19×19 图形）的下一种图形（也是 19×19 的图形）是什么。也就是说，目前状态的棋局是训练集，下一步的棋局形态是训练集的人工标注。我们将 3000 万的人类对弈的位置信息拆解为训练集，反复训练，使网络能尽量和训练集一致，以拟合训练集。

用这个数据集训练后能达到 57%的训练精度。当我第一次看论文时非常惊讶，

这可能吗？一直以为预测人类下棋是一个很难的问题，用传统的卷积网络也无法解决。事实上，卷积网络不但能够预测人类下棋，还能达到如此高的准确度，这也表明了大量的人类棋手对弈时总是凭直觉下棋，而不是像 MCTS 搜索一样，每一步棋都推导到最后。

同时，DeepMind 还对比训练了一个稍小规模的策略网络，它的精度比较低只有 24.2%，但是运行速度却很快，2 微秒就可以运行完毕，不像上一个策略网络需要运行 3 毫秒。实际上这样做的性价比是很高的：精度降低了一半，速度却提高了 1500 倍！

然后，我们用这个训练好的模型预测，这样就可以计算出在某一个棋局状态中的下一个可能落子位置的概率，如图 5-3 所示。

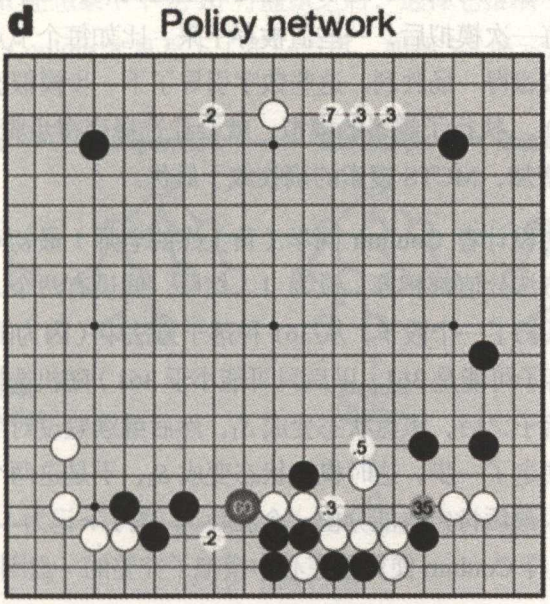


图 5-3 策略网络

而这种方式其实和估值网络的做法一致，只是输出不同，策略网络求的是这个时刻，各个可能的落子位置的概率，估值网络求的是下一个状态的胜利或失败的概率。

这样，就得到了一个策略网络，我们给它起个名字——“P_human”。

P_human 和业余 6 段的人类棋手过招，各有胜负。而当时最强的 AI 叫“CrazyStone”，P_human 无法胜过 CrazyStone，更别提像李世石这样的顶尖人类棋手。

5.3.2 MCTS 拯救了围棋算法

如果我们无法战胜人类最好的棋手，那么我们先战胜最好的 AI 吧，如果要战胜 CrazyStone，先得研究它是如何做到这么好的。

这里要说下 CrazyStone 的主要算法 MCTS。MCTS 搜索是一种搜索博弈树的替代。它的目标是模拟大量的棋局。每一次模拟开始一个棋局，并且在两名选手之一获得胜利后停止棋局模拟。起初，这是完全随机的：对于双方选手来说行为是随机选择的。每一次模拟后，一些值被存下来，比如每个节点多长时间访问一次，多久就会导致赢得一场胜利。这些数字引导了下一步模拟选择行为（模拟因此越来越不随机）。执行了越多的模拟，就会让选择赢得步骤更精确。这表明，随着模拟的数量增加，MCTS 搜索的确收敛于最优。

CrazyStone 的设计者 Coulum 同学（黄士杰的老师）最初对围棋一无所知，便假设所有落子方法分值都相等，设为 1。然后，假设有两个人什么也不懂，开始对弈，其中一人扔了一个骰子，从 361 种落子方法中（因为围棋棋盘是 19×19 的，所以所有的落子可能是 361，以后只可能小于 361）随机选择一个走法 a_0。其中一人想象自己落子之后，棋盘状态变成 S_1，然后继续假设对手也和自己一样扔了一个骰子，随便走了一步，这时棋盘状态变成 S_2，于是这两个人一直扔骰子下棋，一路走到 S_n，最后肯定也能分出一个胜负 r，赢了 r 记为 1，输了则为 0，假设第一次 r=1。这样 Coulum 便算是在心中模拟了完整的一盘围棋。

Coulum 心想，这样随机扔骰子也能赢？运气不错啊，把刚才那个落子方法（S_0,a_0）记下来，分值提高一些：

- 新分数=初始分+r

刚才从（S_0, a_0）开始模拟赢了一次，r=1，那么新分数=2，除了第一步，后

面几步运气也不错，那我把这些随机出的局面所对应的落子方法（S_i, a_i）的分数都设为 2。然后，Coulum 开始做第二次模拟，这次扔骰子时 Coulum 对围棋已经不是一无所知了，但也知道的不是太多，所以这次除（S_0, a_0）的分值是 2 之外，其他落子方法的分数还是 1。再次选择 a_0 的概率要比其他方法高一点。

那位假想中的对手也用同样的方法更新了自己的新分数，他会选择一个 a_1 作为应对。如法炮制，Coulum 又和想象中的对手下了一盘稍微不那么傻的棋，结果它又赢了，于是 Coulum 继续调整它的模拟路径上相应的分数，把它们都+1。随着想象中的棋局下得越来越多，那些看起来不错的落子方案的分数就会越来越高，而落子方案越是有前途，越会被更多地选中进行推演，于是最有“前途”的落子方法就会“涌现”出来。

最后，Coulum 在想象中下完 10 万盘棋之后，选择它推演中分数最高的那个方案落子。这时，Coulum 才真正下了第一步棋，如图 5-4 所示。

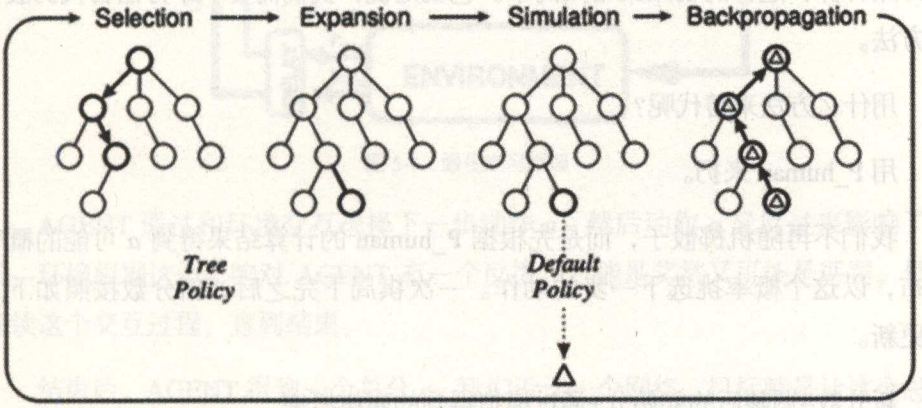


图 5-4 MCTS 过程

MCTS 搜索华丽转身应用到围棋 AI 后，可以看到它有两个很有意思的特点。

（1）没有任何人工的特征，完全依靠规则本身，通过不断想象自对弈来提高能力。这和深蓝战胜卡斯帕罗夫完全不同，深蓝使用了一个复杂的评估函数，这个评估函数是在国际象棋高手帮助下设计出来。而 MCTS 靠的是一种类似遗传算

法的自我进化，只需要遍历树并保持跟踪一些数字，让靠谱的方法自我涌现出来。代价是，必须运行大量模拟以达到良好的效果。

（2）MCTS 可以连续运行，在对手思考对策的同时自己也可以思考对策。Coulum 想象中的两人下完第一步之后，完全不必停下，可以继续进行想象中的对弈，直到对手落子。随后，从对手落子之后的状态开始计算，但是之前的想象中的对弈完全可以保留，因为对手的落子完全可能出现在之前想象的对弈中，所以之前的计算是有用的。这就像人在进行对弈时，可以不断思考，不会因为等待对手行动而中断。这一点，Coulum 的程序非常像人。

最强的围棋 AI 都依赖于 MCTS。它们还依赖于领域知识（由专家设计的手工规则），以便于在蒙特卡洛模拟时能更好地选择行为。这 4 个围棋 AI 都在业余比赛中表现出了强大的水平。

黄士杰很快意识到他老师的程序仍然有局限：起点太低，AI 从完全什么都不会开始计算，这样的初始策略太简单。也就是说，我们需要一个方法替代扔骰子的方法。

用什么方法来替代呢？

用 P_human 来扔。

我们不再随机掷骰子，而是先根据 P_human 的计算结果得到 a 可能的概率分布，以这个概率挑选下一步的动作。一次棋局下完之后，新分数按照如下方式更新。

新分数=调整后的初始分+通过模拟得到的赢棋概率

如果某一步被随机到很多次，就应该主要依据模拟得到的概率而非 P_human。所以 P_human 的初始分会被打个折扣。

调整后的初始分= P_human/（被随机到的次数+ 1）

这样，既可以用 P_human 快速定位比较好的落子方案，又给其他位置一定的

概率。

这一步就是将 MCTS 和之前的策略网络结合。到这里为止，P_human 已经可以战胜已有的围棋 AI 了，接下来它准备挑战人类。

如果靠人类高手和它下棋的方式来提高棋力，显得很不经济，也完全不可行，需要多少次的对弈才能训练出一个具有人类判断能力的围棋 AI 呢？所以我们需要找一个既经济又效率的方法快速提高 AI 棋力——强化学习。

5.3.3 强化学习："周伯通，左右互搏"

我们在本书的第 3 章简单讲了强化学习，先来看看图 5-5。

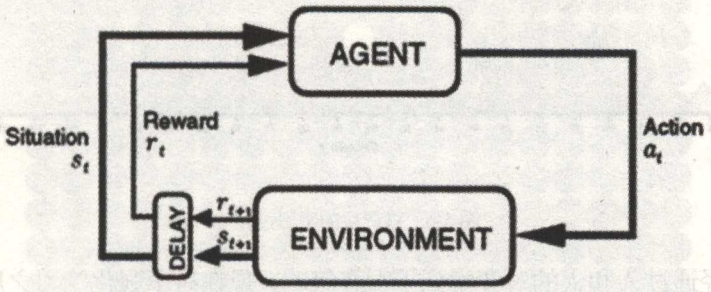


图 5-5 强化学习原理

AGENT 通过和环境交互选择下一步动作 a，然后动作 a 又反过来影响了环境，环境根据这个影响对 AGENT 有一个反馈，可能是奖赏又可能是惩罚，然后继续这个交互过程，直到结束。

结束后，AGENT 得到一个总分 r，我们设计一个网络，目标就是让这个总分变大：将 AGENT 的动作作为训练集，环境返回给他的得分作为标签，这样能用训练集训练一个网络，选择一个单局最高的分数，并且得到这个最高总分情况下 AGENT 的一系列动作。然后再用这些动作重复玩游戏，直到 AI 能得到我们定的最高的那个分数。

仅仅用强化学习能有多厉害？打砖块游戏想必大家都玩过吧？在没有提示时，

人类也会花些时间才能学习到，把球弹到砖块上面能得高分。我们的 AI 通过玩游戏 600 次后，学到了这个诀窍，如图 5-6 所示。

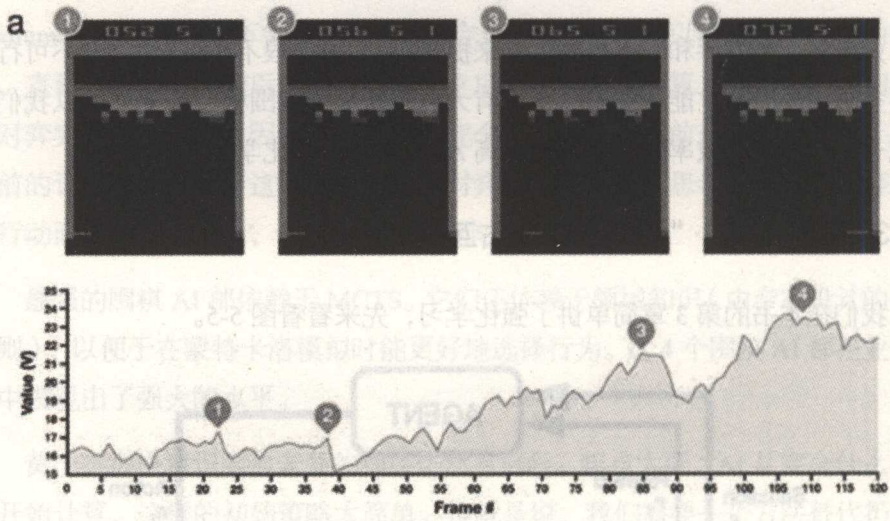


图 5-6　打砖块的学习过程

AI 已经通过人和人的对弈学到了很多策略，现在有了强化学习之后，我们可以让 AI 和自己下棋，然后再用这些棋局去训练网络，这样我们能得到一个近乎无限多的训练集。

当然，这个技术也不是刚出来的，早在 1992 年就被用于 TD-Gammon，IBM 的 Gerald Tesauro 发明了它。TD-Gammon 是一个双陆战棋 AI，那时，它已经能和最棒的人类选手玩得一样好。

然后，DeepMind 团队对网络的性能进行了测试。在每次落子中，它用策略网络来预测赢的概率。使用此策略，每次只用 3 毫秒计算（上面所提到的还没有提高的低精度方法）。我们对比下 Pachi（表现最好的策略网络，最强大的 MCTS 开源项目），在不用任何搜索时，AlphaGo 相比 Pachi 也有 85%的胜率!

卷积神经网络利用强化学习的方式能够超越一个从前表现很好的树搜索。这

再次说明，游戏中直觉是非常重要的。也说明，不用像树搜索那样思考很长时间，不用把每一种可能都考虑到。

5.3.4 估值网络

当策略网络与机器本身对弈时，估值网络开始训练已有的3000万的棋谱位置。这里，估值网络应该按当前游戏状态预测赢的可能性。这类似一个评估函数，不同的是估值网络是学习出来而不是设计出来的，如图5-7所示。

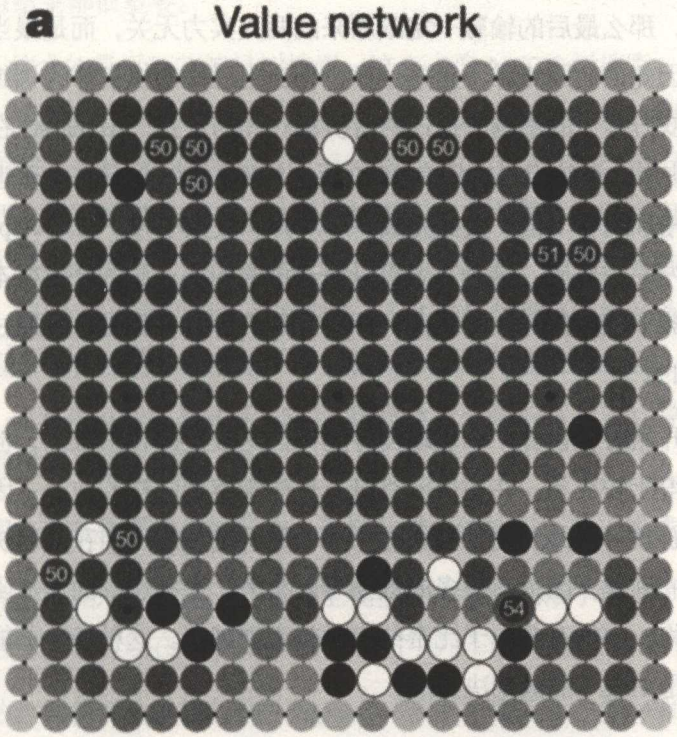


图5-7 估值网络

估值网络关注在目前局势的状况下，每个落子位置的“最后”胜率（这也是所谓的整体棋局），而不是短期的攻城略地。也就是说，策略网络是分类问题（对方会下在哪儿），估值网络是评估问题（我下在这的胜率是多少）。估值网络并不是

一个精确解的评价机制，因为如果要算出精确解可能会耗费极大量的计算能力。因此，它只是一个近似解的网络，即通过卷积神经网络的方式来计算卷积核范围的平均胜率，这个做法的主要目的是将评价函数平滑化，同时，避免过度学习。

当然，这里提到的胜率会跟向后预测的步数有关，向后预测的步数越多，计算就越庞大，AlphaGo 目前有能力自己判断需要展开的预测步数。但是，如何才能确保过去的样本能够正确反映胜率，而且不受对弈双方实力的事前判断（事前判断指的是：判断谁会赢的依据是这个人比较厉害，棋力比较高）？因此，这个部分是通过两台 AlphaGo 对弈的方式来解决的，因为两台 AlphaGo 的实力可以当作是相同的，那么最后的输赢一定跟原来的两人实力无关，而是跟当前落子的位置有关。也因此估值网络并不是透过这个世界上已知的棋谱作为训练，因为人类对弈会受到双方实力的影响。透过两台机器对弈的方式，AlphaGo 在与欧洲棋王对弈时，所使用的训练组样本只有 3000 万个棋谱位置，但是在与李世石比赛时棋谱位置已经增加到了 1 亿。人类完成一局对弈一般数小时，但是 AlphaGo 间对弈可能在一秒内完成数局，这种方式可以快速地累积出正确的评价样本。所以，先前提到的机器下围棋的最大困难点：评价机制的部分，就是这样通过卷积神经网络来解决掉的。

在比赛开始时，设计一个表现良好的评估函数是比较困难的，但在接近对弈结束时则变得比较容易。在估值网络身上也能观察到同样的效果：因为没有数值可以参考，游戏一开始，估值网络做了一个随机的预测，但在游戏快终结时更善于预测更多的动作，因为有更多的数据可以参考。人类设计的评估函数和通过学习的估值网络，都能在趋向于比赛结束时越表现越好。这种趋势不能归于人类的局限性，而是由根本的规则决定的。

5.3.5 将所有组合到一起：树搜索

我们将上面训练好的 P_human、走棋策略网络、估值网络组合起来，保证离线训练的部分和在线决策及估算部分能够整合起来，并把它放到真实的对弈环境中运行，这就是 AlphaGo。

AlphaGo 正是靠树搜索将三种类型的网络以一种全新的方式整合起来。它所使用的树搜索有点像 MCTS。

AlphaGo 使用一个评估函数给定状态的估算值，即估值网络输出值与策略网络的混合体：状态的值=估值网络输出值+模拟结果。这有点类似于深蓝。

仔细想想，这看起来很像我们人类的思维，是直觉和反射的混合体。估值网络提供直觉，而模拟结果提供了条件反射。AlphaGo 团队也尝试只使用估值网络的输出，或者只有模拟结果输出，但这些比两者的结合要糟糕。所以，估值网络的输出和模拟结果都很重要。

AlphaGo 技术的最后环节就是树搜索，这有点像 MCTS 树搜索，它将三种类型的网络以一种创新的方式引导在一起。相较于以前深蓝所使用的搜索（主要是 MinMax 搜索算法，这里就不再赘述），由于我们并不具有无限大的计算能力（请注意，如果是有限的排列组合，MCTS 树搜索的确有可能针对所有组合进行通盘评估，但是在围棋的场景下是没有办法的，就算这样做，恐怕也会造成计算时间的大幅增加），因此不可能适用于旧的方法，不过在前面策略网络及估值网络中，AlphaGo 已经可以针对接下来的落子（包括对方）将可能性缩小到一个可控的范围。接下来，它就可以快速地运用 MCTS 树搜索在有限的组合中计算最佳解。一般来说树搜索包括 4 个步骤。

（1）选取：首先根据目前的状态，选择几种可能的对手落子模式。

（2）展开：根据对手的落子，展开至我们胜率最大的落子模式（我们称之为一阶蒙特卡洛树），所以在 AlphaGo 的搜寻树中并不会真的展开所有组合。

（3）评估：如何评估最佳行动（AlphaGo 该下在哪儿），第一种方式是将行动后的棋局丢到估值网络中来评估胜率，第二种方式是做更深度的 MCTS（多预测几阶可能的结果）。这两种方法所评估的结果可能截然不同，AlphaGo 使用了混合系数（Mixing Coefficient）将两种评估结果整合，目前在 Nature 刊出的混合系数是 50%-50%（但是我猜实际一定不是）。

（4）后向传递：在决定我们最佳行动位置后，快速根据这个位置向下透过策

略网络评估对手可能的下一步，以及对应的搜索评估。所以，AlphaGo 其实最恐怖的是，李世石在思考自己该下哪里的时候，不但 AlphaGo 可能早就猜出了他可能落子位置，而且正利用他在思考的时间继续向下计算后面的棋路，如图 5-8 所示。

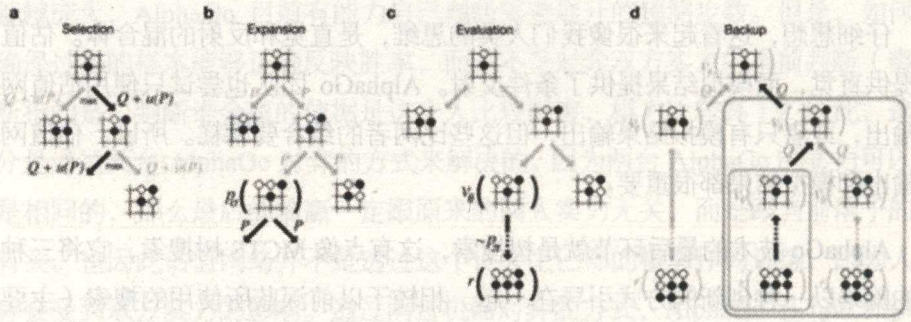


图 5-8　树搜索

根据 AlphaGo 团队的实测，单机版的 AlphaGo 如果单独使用一个大脑或是蒙利卡罗搜索树技术，都能达到业余（段）的等级（欧洲棋王樊麾实力等级大概是在 2500~2600，而李世石是在 3500 以上）。但是，当这些技术整合时，就能呈现更强大的力量。在刊登 Nature 上的论文时，它的预估强度大概也只有职业 3~4 段（李世石是 9 段），不过刚刚提到它透过强化学习技术增强策略网络、透过两台 AlphaGo 优化估值网络，这些都可以让他在短时间内变得更强大。而且，电脑没有情感也不怕压力，更不会因为对手的表现而轻敌（AlphaGo 的策略网络一向只预测强者），人类就算有更强大的实力也可能因为无法承受输赢压力而不能发挥最好水平。图 5-9 所示为几种技术的围棋等级分。

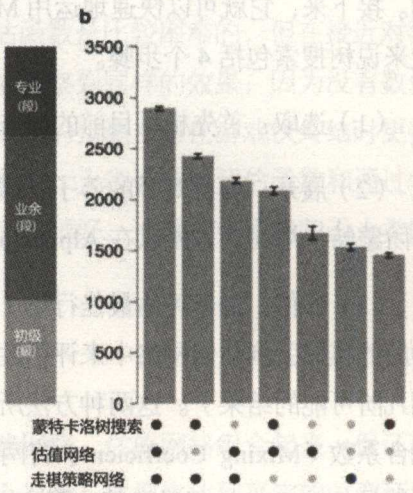


图 5-9　几种技术的围棋等级分

5.3.6 AlphaGo 有多好

AlphaGo 正在和李世石对弈，如图 5-10 所示。

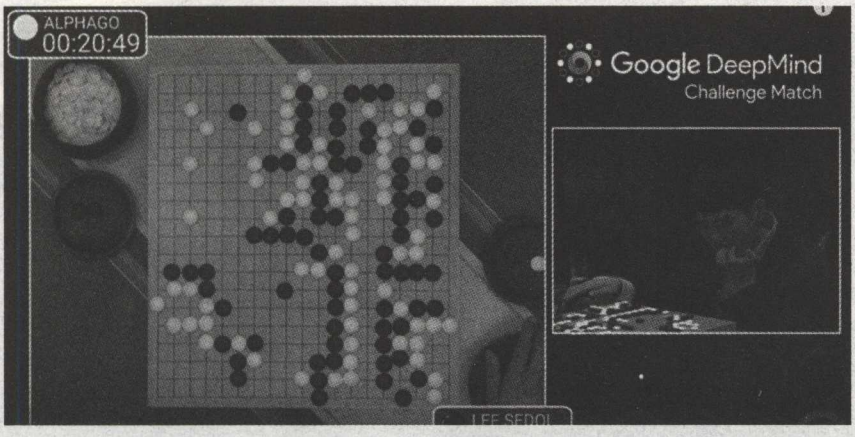


图 5.10 AlphaGo 正在和李世石对弈（新闻图片）

AlphaGo 相比其他人类和人工智能有多强？最常用来比较选手实力的系统叫国际等级分（Elo Rating）。用两个选手之间的等级差异作为预测比赛的结果，更高的评级表明更高的获胜机会。

在 2015 年的最新文章里，各种 AI 的实力估计如表 5-1 所示。

表 5–1 各种 AI 的实力估计

AI 名称	国际等级分
Distributed AlphaGo (2015)	3140
AlphaGo (2015)	2890
CrazyStone	1929
Zen	1888
Pachi	1298
Fuego	1148
GnuGo	431

（数据来自 Go Ratings 网站）

单机版的 AlphaGo 跑在 48 个 CPU 和 8 个 GPU 上，AlphaGo 的分布式版本跑在 1202 个 CPU 和 176 个 GPU 上，这就是它大致的规模，如图 5-11 所示。

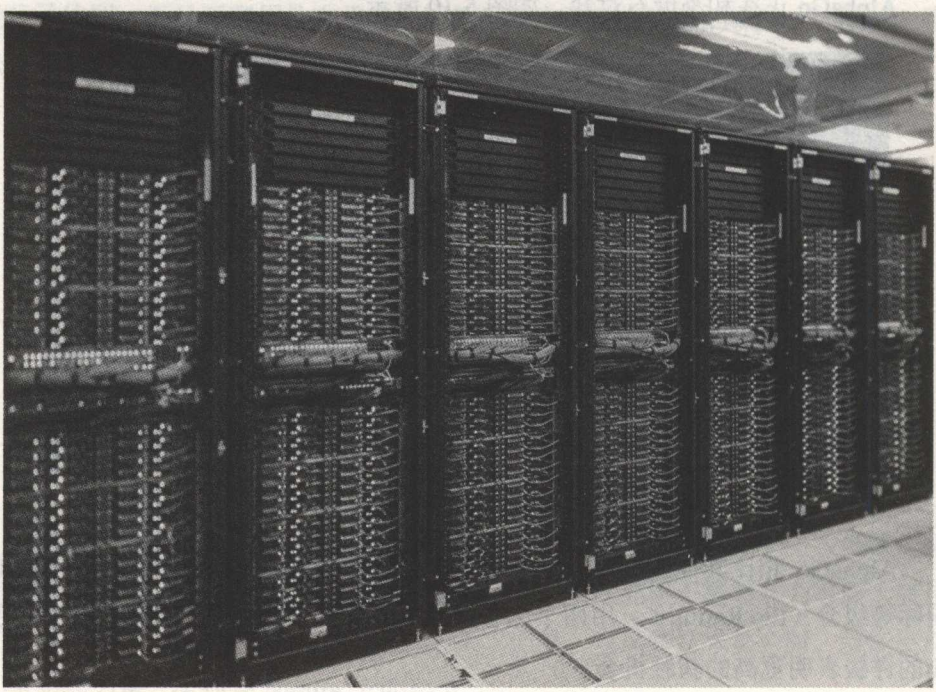


图 5-11　分布式的 AlphaGo

我们看到更多的计算资源可以带来更好的游戏性能，但国际等级分没有估计 AlphaGo 在单个 CPU 上运行的性能。

2015 年 10 月，分布式版本的 AlphaGo 已经在五局赛制中击败了樊麾。樊麾是专业 2 段选手，并且他在国际等级分中有 2908 的评分。

2016 年 3 月 15 日，分布式版本的 AlphaGo 在对弈李世石时以 4 : 1 的比分击败李世石，而李世石的国际等级分为 3520，分布式版本的 AlphaGo 现在的等级分为 3586。如果从 2015 年起没有改进的话，AlphaGo 不可能击败李世石。

现在，人类棋手中只有一人的分数高于 AlphaGo（柯洁，分数 3621）。

2016 年 5 月最新国际等级分值如图 5-12 所示。

Rank	Name	♂♀	Flag	Elo
1	Ke Jie	♂		3625
2	Google DeepMind AlphaGo			3600
3	Park Junghwan	♂		3558
4	Lee Sedol	♂		3539
5	Shi Yue	♂		3535
6	Iyama Yuta	♂		3528
7	Kim Jiseok	♂		3521
8	Zhou Ruiyang	♂		3507
9	Lian Xiao	♂		3506
10	Park Yeonghun	♂		3504

图 5-12　2016 年 5 月最新国际等级分值

5.3.7　总结

AlphaGo 总体上包含离线学习和在线对弈两个过程。

离线学习过程分为三个训练阶段。

第一阶段：利用 3000 万专业棋手棋谱位置来训练两个网络。一个是基于全局特征和 CNN 训练出来的走棋策略网络，其主要作用是给定当前盘面状态作为输入，输出下一步棋在棋盘其他空地上的落子概率。另一个是利用局部特征和线性模型训练出的快速走棋策略。策略网络速度较慢，但精度较高；快速走棋策略反之。

第二阶段：利用第 t 轮的策略网络与先前训练好的策略网络互相对弈，利用强化学习来修正第 t 轮的策略网络的参数，最终得到增强的策略网络。这部分应该存在理论上的瓶颈（提升能力有限）。

第三阶段：先利用普通的策略网络生成棋局的前 U-1 步（U 是一个属于[1, 450]的随机变量），然后利用随机采样决定第 U 步的位置（这是为了增加棋的多样性，防止过拟合）。随后，利用增强的策略网络完成后面的自我对弈过程，直至棋局结束分出胜负。此后，第 U 步的盘面作为特征输入，胜负作为 label，学习一个估值网络，用于判断结果的输赢概率。估值网络其实是 AlphaGo 的一大创新，围棋最为困难的地方在于很难根据当前的局势判断最后的结果，这点职业棋手也很难掌握。通过大量的自我对弈，AlphaGo 用了 3000 万棋局位置，用来训练估值网络。但由于围棋的搜索空间太大，3000 万棋局位置也不能帮 AlphaGo 完全克服

这个问题。

离线学习后，AlphaGo 进入在线对弈阶段。在线对弈过程包括以下 5 个步骤：其核心思想是在 MCTS 中嵌入深度神经网络来减少搜索空间。

根据当前盘面已经落子的情况提取相应特征。利用策略网络估计出棋盘其他空地的落子概率。

根据落子概率计算此处往下发展的权重，初始值为落子概率本身（如 0.18）。实际情况可能是一个以概率值为输入的函数，此处是为了理解方便。利用估值网络和快速走棋网络分别判断局势，两个局势得分相加为此处最后走棋获胜的得分。这里使用快速走棋策略是一个用速度来换取量的方法，从被判断的位置出发，快速行棋至最后，每一次行棋结束后都会有个输赢结果，然后综合统计这个节点对应的胜率。而估值网络只要根据当前的状态便可直接评估出最后的结果。两者各有优缺点，可以互补。

利用第四步计算的得分来更新之前那个走棋位置的权重（如从 0.18 变成了 0.12）。此后，从权重最大的 0.15 那条边开始继续搜索和更新。这些权重的更新过程应该是可以并行的。当某个节点的被访问次数超过了一定的门限值，则在蒙特卡洛搜索树上进一步展开下一级别的搜索，如图 5-13 所示。

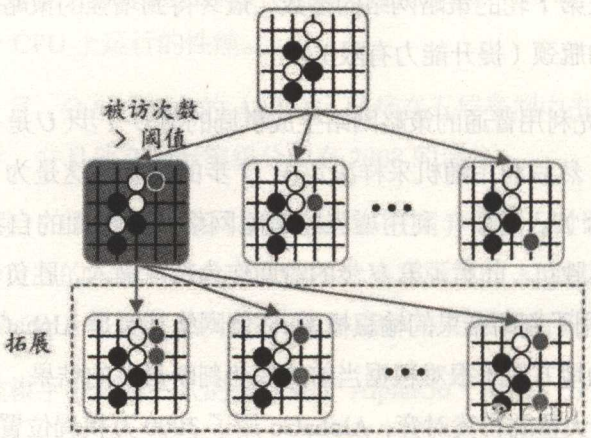


图 5-13　MCTS

5.4 重要的技术进步

从 AlphaGo 的胜利中至少可以产生两个重要的影响。第一是技术上的，和本次成就的地位相似的有：深蓝赢得了国际象棋比赛的胜利；沃森的一系列成就。第二是文化上的，普通大众不会有什么特别的感觉，但这一事件将被历史学家和人工智能专家视为一个重要的里程碑。这里要提下我现在的团队，在 AlphaGo 对弈的几天时间里，我们团队成员对于比赛结果非常期待，甚至到了对输赢互相打赌的地步。

哪些部分让人印象深刻，哪些进步意义重大?

首先，AlphaGo 并不像深蓝那样评估数以百万计的位置（AlphaGo 每次都只会评估上千次位置），而是选择位置评估，然后仔细思考。这是更类似于人类的思考过程。

从这个角度来看，搜索算法可能并不那么重要。从技术上讲，改进这个算法是比较难的，但在完全信息的博弈树中搜索却很好理解，尤其是蒙特卡洛树搜索已经应用在围棋上了。

与搜索算法相比，更重要的是策略网络，它能预测下一步落子。这对于有更多选项的围棋来说是至关重要的，对国际象棋也一样。

决定哪些步骤需要思考是一个普遍存在的问题，不仅在游戏中，在生活中也一样。这在计算机建模中一直是很困难的——人类使用启发、直觉和经验的混合体生成一组选项，只有经验真的适合大部分人工智能技术。

策略网络部分解决了这个问题。DeepMind 深度神经网络可以比人类专家做更好的预测，而自我对弈能大大增加它们预测落子的能力。

1. 深度神经网络现在能做到的事情，比以前的任何技术更像人类

如果它没有了人类专家的观察、记录和引导。它能真的完成整个学习吗?我认

为是可能的，而且 DeepMind 可能试图这样做。

估值网络是 AlphaGo 最令人印象深刻的技术，它可以观察棋局并且估计谁赢谁输。对棋局结果的预测其实是非常困难的，围棋 9 段评论员和大部分棋手以及爱好者观看了比赛，在棋局进行到一半时往往也不能真正知道谁将获胜，甚至错误地认为 AlphaGo 会失败。

2. 不可思议的近似人类的直觉，将改变游戏规则

生活中，我们决定一件事是完全不透明的。有时从直觉上，我们会做出依赖情绪的决定，然后使用我们的有意识的头脑合理化决定或试图去解释做出选择的原因。这听起来好像很傻，但在自然界中，人类大脑进化到今天已经非常成功了。

这使得直觉很难用在其他计算机模型里。训练后的 AlphaGo 或许有看起来很像人类直觉的东西，但它只能专注于一个特定的比赛，不能同时覆盖其他领域。

5.5 一些可以改进的地方

从表面上看，围棋和国际象棋都代表了典型的、我们所面临的人工智能的挑战：一是决策的任务是具有挑战性的，二是搜索空间很庞大。

国际象棋可以用一个相对简单的解决方案击败最好的人类玩家：蛮力搜索，加上一些象棋高手启发式的手工规则。启发式的规则系统必须用手工编写，这很令人失望，因为它无法引导 AI 的突破，因为每个新问题都需要编写新的手工规则。深蓝的发明者很幸运，虽然国际象棋的状态空间很大，但仍然在人类可以处理的范围内。

而围棋，由于它的高复杂性，用深蓝系统的方法无法解决。幸运的是，AlphaGo 使用了 MCTS。但纯 MCTS 没有任何领域知识，这意味着一个围棋程序如果只使用 MCTS，在每个新棋局开始时，程序根本不知道该如何走下去，它没有学习经

验的能力。

AlphaGo 有趣的地方在于，它包含一个类似可学习新规则的组件来代替手工编写启发式规则。通过与自己对弈，AlphaGo 在围棋领域变得越来越好，这一方法可能对其他人工智能领域的进步有帮助。

同样有趣的是，尽管 AlphaGo 的计算能力比深蓝强得多，但 AlphaGo 的评估位置比深蓝少数千倍。由于 AlphaGo 使用策略和估值网络，因此它的位置的评估更准确，可以得到更好的比赛结果。

AlphaGo 对李世石的比赛严重依赖了庞大的计算能力。如果只有很少的 CPU 和 GPU 可用，那比赛结果就不好说了。

AlphaGo 做得并不完美，未来仍有一些改进空间。

- AlphaGo 依赖人类选手比赛的大型数据集。幸运的是，我们已经有一个人类高手比赛的大数据集，但在其他人工智能问题上就可能没有这个条件了。因此 AlphaGo 可能有点偏向于模仿人类，如果只依赖自我对弈，也许能发现完全相反的新的策略。TD-Gammon 就是如此，它只依赖自我对弈。根据维基百科：[TD-Gammon]探索策略以一种人类没有尝试过并能得到正确结果的方式进行。
- 系统不是端到端的被训练。AlphaGo 有两个不同的部分：神经网络和树搜索阶段。目前神经网络和树搜索阶段是独立训练的，如果把它们联合学习，也许会得到一个更好的学习-搜索过程。
- AlphaGo 依赖卷积网络（上文我们已经提到），这是策略和估值网络的基础结构。利用卷积网络不可能解决未来所有的比赛或人工智能问题。
- 很多超参数需要设置。如果系统使用的很多参数大部分可以在对弈中形成，那么先验信息可能也不是很必要；如果这些参数的获取也能通过自动学习获得，而不必手动设置，那就更好了（当然，前提是得到更好的比赛结果）。

5.6 未来

有人说，如果用一个 AI 能够解决一个特定的任务或者获得一个游戏的胜利，这将是真正智能的标志。

在跳棋、象棋、医学诊断领域，这类 AI 都渐渐出现了；而且，我们知道，暂时不会出现一种通用的 AI 满足所有的领域所需，这次对弈李世石的胜利只是宣称“仅仅”解决了围棋的问题。但这次胜利不是通过预设一些规则，让 AI 条件反射地做事情。所以，这才是真正的人工智能的方向。

AlphaGo 标志了神经网络（深度学习）和强化学习开始应用在人类级别的 AI 中，从这个角度来说，这场胜利确实是一个里程碑。

同样，这对短期技术进步会有巨大的影响。运用神经网络即将或正在创造一些非常有用的东西，比如智慧助手、自动驾驶，理解广告向我们展示些什么，预测人们会购买什么产品，等等。

机器学习和人工智能将用它的方式改变这个社会。最终，我们会有一个像人类一样思考的机器，甚至比人类更好。

未来，机器 AI 会替代人类的工作吗？重复性质的工作首先会被替代，而一些表达人类情感和艺术类的工作呢？我认为 AI 能产生比人类更有想象力或更有创造力的作品，某种意义上说，甚至比人类的作品更好。差别在于人类的作品能和人类自己保持更深层次的情感共鸣，或者摸到人类更深层的脾气，更容易贴近人类的某一类情感需求。这会导致艺术家的工作内容发生转变，虽然还叫艺术家，但已经不独立作画或写作了，更有可能是借助 AI 完成作品。

另外，人类的直觉是很重要的东西。无论是工作还是生活，人类靠直觉处理很多事情，而 AI 直觉有可能依靠大规模训练和结构的进化在某一天超过人类。当然，到那个时候，机器的直觉或者思维人类可能无法理解，人类真正创造了另一“物种”。

最后，类似 DeepMind 这样的公司能继续领导下去吗?我更愿意相信一个肯定的答案，尤其是在 DeepMind 已经创造了一个人类能力所不能及的围棋 AI 之后。

6 两个重要的概念

6.1 迁移学习

迁移学习（Transfer Learning）在最近几年引起了广泛的关注和研究。根据维基百科的定义，迁移学习是运用已有的知识对不同但相关领域问题进行求解的一种新的机器学习方法，迁移学习的目标是将从一个环境中学到的知识用来帮助新环境中的学习任务，迁移已有的知识解决目标领域中仅有少量有标签样本数据甚至没有样本数据的学习问题。迁移学习在人类的大脑思考中广泛存在，两个不同领域相关的方面越多，迁移学习就越容易实现；相关的方面越少，则迁移学习就越困难，甚至可能产生副作用。例如，学会了轮滑的人，就很容易学会滑冰刀，但是有时看起来很相似的事情，却有可能产生“负迁移”现象。比如，习惯打羽毛球的人学习打网球时可能反倒会走弯路，因为发力的位置不同。

我们先来看看人类在学习新知识时是怎么运用已知知识的，如图 6-1 所示。

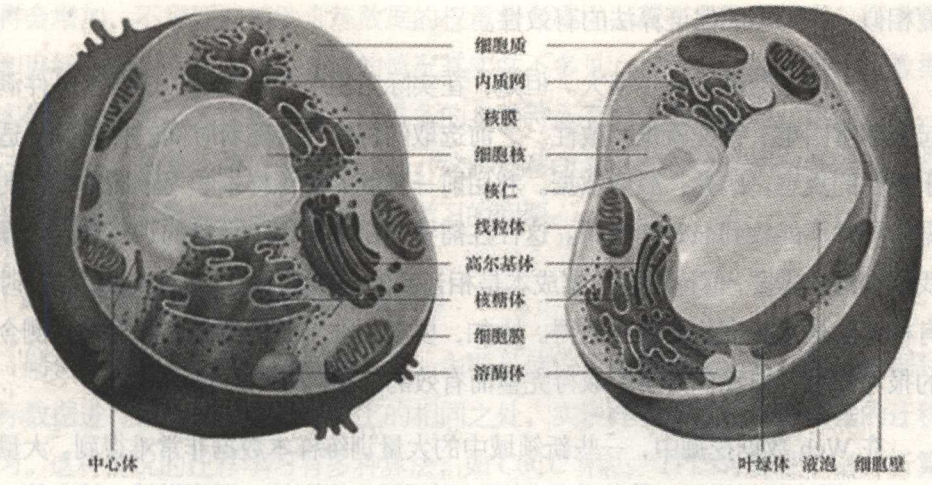


图 6-1 左边是动物细胞结构图，右边是植物细胞结构图

假设小明学习了什么是植物细胞，学习了什么是细胞质、细胞核、线粒体、细胞膜等，知道了植物的光合作用是怎么发生的。现在，老师开始教授动物细胞知识，还需要从头到尾再讲一遍细胞质、细胞核和线粒体吗？聪明的老师肯定会用植物细胞举例，表明动植物细胞的异同点，通过细胞结构图清楚地看到，植物细胞具有的结构动物细胞大部分都有，但有些地方还是存在差异，比如最大的区别是植物细胞有细胞壁而动物细胞只有细胞膜——这是区分动植物细胞的显著特征。

小明开始学习植物细胞后，接着再学习动物细胞结构图，也不需要再次将所有的细胞结构学习一次，只要跟着老师的思路查看细胞结构的异同点，了解动物细胞有什么不同，为什么会造成这些不同，就能很好地了解动物细胞的特点。

小明在学习动物细胞的过程中，利用学习植物细胞的知识点，加速完成整个动物细胞的学习过程就是迁移学习。

回过头来我们看看在机器学习中需要什么条件来满足学习。在传统分类学习中，为了保证训练得到的分类模型准确可靠，所选取的样本都基于以下两点假设。

（1）测试样本和训练样本独立分布，也就是说，样本必须和实际应用场景高

度相似，否则不能保证算法的有效性。

（2）样本数量必须足够大。但是，在实际应用中我们发现，这两个条件满足起来非常困难。首先就是时效性。之前选取的样本可能随着时间的推移并不适用于现在的推理。比如，股票数据。利用前一年度的训练样本学习预测本年度的数据可能并不能收到很好的效果。这往往需要我们重新标注大量的训练数据以满足我们训练的需要，但标注新数据成本是相当高的，需要大量的人力与物力。同时，有标签的样本数据往往很难获得。比如，要根据往年的高考报名数据来预测今年的报考数据，但我们并不能取得完整而有效的往年的报考数据样本。

在 Web 数据挖掘中，一些新领域中的大量训练样本数据非常难得到。大量新的 Web 领域不断涌现，从传统的新闻，到网页，到图片，再到微博、博客等。传统的机器学习需要对每个领域都标注大量样本数据，而标注大量的样本数据又非常费时费力。如果缺少大量标注的样本数据，那么很多与机器学习相关的研究与应用都无法开展。为了机器学习，我们要花费人力物力不断地标注大量新的样本数据；同时，对于已有的样本数据，完全丢弃也是非常浪费的。如何合理地利用这些数据是迁移学习主要解决的问题。迁移学习可以从现有的样本数据中迁移知识到相关领域，用来帮助相关新领域的学习。利用好迁移学习，可以提高样本数据利用率，提高机器学习的效率。

我们在迁移学习方面的工作目前可以分为以下三个部分：同构空间下基于实例的迁移学习，同构空间下基于特征的迁移学习与异构空间下的迁移学习。其中，基于实例的迁移学习知识迁移能力更强，基于特征的迁移学习知识迁移能力更广泛，而异构空间的迁移具有广泛的学习与扩展能力。这几种方法各有不同的优势，下面我们具体谈一下。

1. 同构空间下基于实例的迁移学习

基于实例的迁移学习应用范围较窄，仅可应用在样本数据与目标数据非常相近的情况下。在这种情况下，常用的一种算法是具有迁移能力的 boosting 算法。boosting 的作用是建立一种自动调整权重的机制，于是重要的辅助训练数据的权

重将会增加，不重要的辅助训练数据的权重将会减小。而在实际应用中，样本数据和目标数据在实例上交集较多的情况其实并不多见，有时样本数据与目标数据虽实例上并没有很多交集，但它们具备的主要特征有较大的相关性。因此还有一类迁移学习，是基于特征的迁移学习，这种迁移学习主要是如何将特征层面上具有相关性的样本数据和目标数据进行学习的问题。

2. 同构空间下基于特征的迁移学习

同构空间下基于特征的迁移学习，主要是使用互聚类算法同时对样本数据和目标数据进行聚类，寻找到特征上的相同之处，实现样本数据到目标数据的迁移学习。这种情况的迁移学习有多种算法，如 CoCC 算法、TPLSA 算法、谱分析算法与自学习算法等。

以上两种学习方法解决的都是样本数据与目标数据在同一特征空间内时的迁移学习问题。很多情况下，样本数据与目标数据处于不同的特征空间中，此时的学习就是跨特征空间的迁移学习，也就是异构空间下的迁移学习。

3. 异构空间下的迁移学习：翻译学习

当样本数据和目标数据分别属于两个不同的特征空间时，我们可以使用异构空间下的迁移学习方法：翻译学习。我们找到样本数据与目标数据之间的某种桥梁关系，从而构建一个翻译器，使这两种本不属于一个空间下的数据可以利用翻译器，将样本数据翻译到目标数据的特征空间中去，进行学习和分类。

6.2 概率图模型

首先，大家要了解概率图是什么？其实概率图简单说来就是概率论和图论的结合，科学家为了方便好记，也为体现它的特点取名概率图模型（PGM：Probabilistic Graphical Model）。

如何将概率论和图论结合呢？一般来说，我们将在概率模型上，使用图的方

法表示概率分布（能量函数、密度函数），这是对没有被确定的知识（或预测模型或分类模型）的一种表示方法。

图 6-2 所示为概率图。

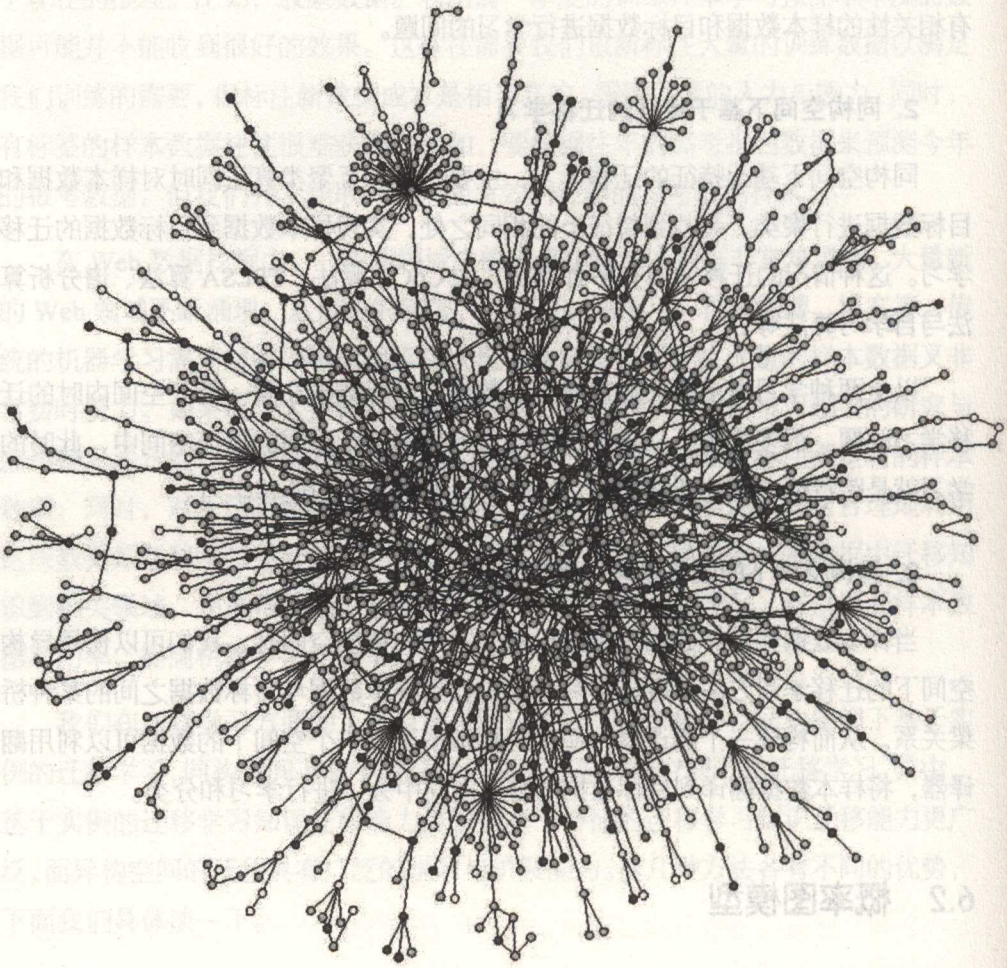


图 6-2　概率图

这是概率图？！不学了吧，谁看得懂？其实这图我也看不懂。我们还是简单点吧，看看图 6-3。

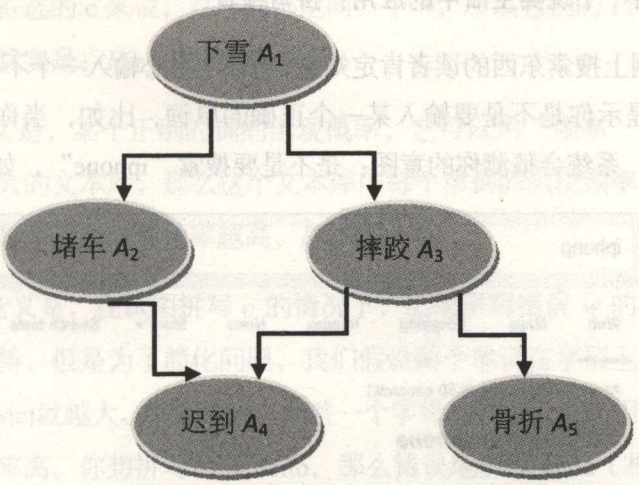


图 6-3 简单的概率图

下面我们从最简单的贝叶斯网络结构入手学习概率图模型。

学习贝叶斯网络之前我们需要简单了解贝叶斯方法。

贝叶斯方法的提出

比如，山上有一家狼，每天有只狼下山去抓羊，你明明知道抓羊的结果就两种，即要么成功要么失败，但你依然会忍不住去计算下狼兄成功的概率有多大。如果你对狼的能力比较了解，若它是一只有方法、思路清晰、有毅力且能利用周围的一切工具和环境（这是不是有点像灰太狼同学）的狼，你会不由自主地估计它成功的概率可能在 80%以上。这种不同于最开始的“非黑即白、非 0 即 1”的思考方式，便是贝叶斯式的思考方式。

下面是根据贝叶斯定理推导出的公式。

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

从这个公式可以看出，与条件概率相似，我们可以用条件概率直接推导出贝叶斯公式。即 $P(A,B) = P(A)P(B|A) = P(B)P(A|B)$，所以 $P(A|B) = P(A)P(B|A) / P(B)$。

我们来看一个现实生活中的应用：拼写检查

经常在网上搜索东西的读者肯定知道，当你不小心输入一个不存在的单词时，搜索引擎会提示你是不是要输入某一个正确的单词，比如，当你在谷歌中输入“iphonp”时，系统会猜测你的意图：是不是要搜索“iphone”，如图 6-4 所示。

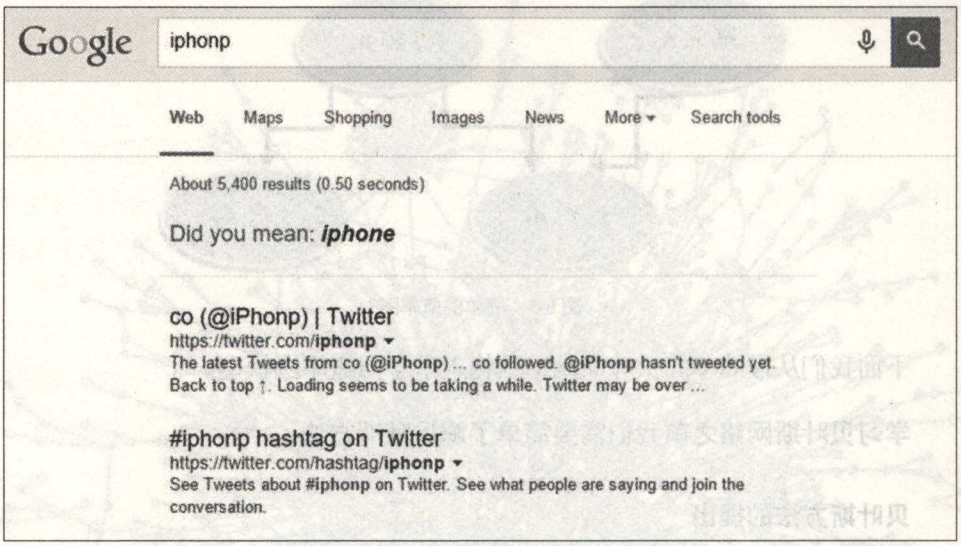


图 6-4　谷歌的纠错能力

这叫拼写检查。根据谷歌一名员工写的文章，Google 的拼写检查主要是基于贝叶斯方法。下面我们就来看看，怎么利用贝叶斯方法，实现“拼写检查”的功能。

用户输入一个单词时，可能拼写正确，也可能拼写错误。如果把拼写正确的情况记作 c（代表 correct），拼写错误的情况记作 w（代表 wrong），那么“拼写检查”要做的事情就是：在发生 w 的情况下，试图推断出 c。从概率论的角度看，就是已知 w，在若干个方案中选择可能性最大的那个。这就完全符合贝叶斯公式要解决的问题：推断出一个概率，也就是求 $P(c|w)$的最大值。

根据贝叶斯定理：

$$P(c|w) = P(w|c)\quad P(c) / P(w)$$

对于所有备选的 c 来说，对应的都是同一个 w，所以它们的 $P(w)$是相同的，因此我们求的其实是：$P(w|c)P(c)$的最大值。

$P(c)$的含义是，某个正确的词的出现概率，它可以用“频率”代替。如果我们有一个足够大的文本库，那么这个文本库中每个单词的出现频率，就相当于它的发生概率。某个词的出现频率越高，$P(c)$就越大。

$P(w|c)$的含义是，在试图拼写 c 的情况下，出现拼写错误 w 的概率。这需要统计数据的支持，但是为了简化问题，我们假设两个单词在字形上越接近，就越可能拼错，$P(w|c)$就越大。举例来说，相差一个字母的拼法，就比相差两个字母的拼法的发生概率高。你想拼写单词 hello，那么错误地拼成 hallo（相差一个字母）的可能性，就比拼成 haallo 高（相差两个字母）。

所以，我们只要找到与输入单词在字形上最相近的那些词，再在其中挑出出现频率最高的一个，就能实现 $P(w|c)\ P(c)$的最大值。

下面我们看看贝叶斯网络的相关概念。

6.2.1 贝叶斯的网络结构

贝叶斯网的网络结构是一个有向无环图（Directed Acyclic Graph），其中每个节点代表一个属性或者数据变量，节点间的弧代表属性（数据变量）间的概率依赖关系。一条弧由一个属性（数据变量）A 指向另外一个属性（数据变量）B，说明属性 A 的取值可以对属性 B 的取值产生影响，由于是有向无环图，A 和 B 间不会出现有向回路。在贝叶斯网中，直接的原因节点（弧尾）A 叫作其结果节点（弧头）B 的双亲节点（parents），B 叫作 A 的孩子节点（children）。如果从一个节点 X 有一条有向通路指向 Y，则称节点 X 为节点 Y 的祖先（ancestor），同时称节点 Y 为节点 X 的后代（descendent）。

我们用下面的例子具体说明贝叶斯网的结构，如图 6-3 所示。

图中共有5个节点和5条弧。下雪 A_1 是一个可以观测的事实，也是原因节点，它会导致堵车 A_2 和摔跤 A_3，而我们知道堵车 A_2 和摔跤 A_3 都可能最终导致上班迟到 A_4。另外，如果在路上摔跤严重的话还可能导致骨折 A_5。这是一个简单的贝叶斯网络的例子。在贝叶斯网中像 A_1 这样没有输入的节点被称为根节点（root），其他节点统称为非根节点。

贝叶斯网络当中的弧表达了节点间的依赖关系，如果两个节点间有弧连接说明两者之间有因果联系，反之如果两者之间没有直接的弧连接或者是间接的有向联通路径，则说明两者之间没有依赖关系，是相互独立的。节点间的相互独立关系是贝叶斯网络中很重要的一个属性，可以大大减少建网过程中的计算量，同时根据独立关系来学习贝叶斯网络也是一个重要的方法，这在本文后面会着重介绍。使用贝叶斯网络结构可以使人得出属性节点间的关系，进而使得使用贝叶斯网进行推理和预测变得相对容易实现。

从图中我们可以看出，节点间的有向路径可以不止一条，一个祖先节点可以通过不同的途径影响它的后代节点。如我们说下雪可能导致迟到，而导致迟到的直接原因可能是堵车，也可能是在雪天滑倒、摔了一跤。每当我们说一个原因节点的出现会导致某个结果的产生时，都是一个概率的表述，而不是必然的，这就需要为每个节点添加一个条件概率。一个节点在其双亲节点（直接的原因接点）的不同取值组合条件下取不同属性值的概率，就构成了该节点的条件概率表。

1. 条件概率表

我们已经介绍了条件概率的概念，贝叶斯网络中的条件概率表是节点的条件概率的集合。当使用贝叶斯网络进行推理时，实际上是使用条件概率表中的先验概率和已知的证据节点来计算所查询的目标节点的后验概率的过程。

条件概率可以由某方面的专家总结以往的经验给出（但这是非常困难的，只适合某些特殊领域），另外一种方法就是通过条件概率公式在大样本数据中统计

求得，学习条件概率表的算法将在后文中详细介绍。在这里我们先根据图 6-4 所示的贝叶斯网给出其中的一些条件概率表，使大家对条件概率表有一个感性的认识。

如果将节点下雪 A_1 当作证据节点，那么发生堵车 A_2 的概率多大呢？表 6-1 给出了相应的条件概率。

表 6–1　发生的概率

A_1	$P(A_2\|A_1)$	
	True	False
True	0.8	0.2
False	0.1	0.9

上表是最简单的情况，如果有不止一个双亲节点的话，那么情况会变得更复杂，如表 6-2 所示。

表 6–2　发生的概率

A_2	A_3	$P(A_4\|A_2, A_3)$	
		True	False
True	True	0.9	0.1
True	False	0.8	0.2

由表可以看出，当堵车 A_2 和摔跤 A_3 取不同属性值时，导致迟到 A_4 的概率不同。贝叶斯网条件概率表中的每个条件概率都是以当前节点的双亲节点作为条件集的。如果一个节点有 n 个父节点，在最简单的情况下（即每个节点都是二值节点，只有两个可能的属性值：True 或者 False），它的条件概率表有 $2n$ 行；如果每个属性节点有 k 个属性值，则有 k_n 行记录，其中每行有 k–1 项（因为 k 项概率的总和为 1，所以只须知道其中的 k–1 项，最后一项可以用减法求得），这样该条件概率表将一共有$(k-1)\,k_n$ 项记录。

根据条件概率和贝叶斯网络结构，我们不仅可以由祖先节点推出后代的结果，

还可以通过后代当中的证据节点向前推出祖先取各种状态的概率。

看到这里是不是觉得概率图很强大?

（1）条件和结果概率是可以互相转化的。

（2）只要已知事件发生的一个或几个分支条件，我们就可以计算出其他分支或结果的概率。

6.2.2 概率图分类

不同的概率图的模型可以分成 3 类：有向图概率模型，无向图概率模型和混合概率图模型（混合概率图本书不做讨论）。

1. 有向图概率模型

有向图概率模型使用有向边连接不同的节点，这些有向边通常表示了节点间的因果关系。典型的代表是隐马尔可夫模型（HMM，NewRoman），贝叶斯网络和动态贝叶斯网络。

HMM

看这个名称就知道，隐马尔可夫模型是马尔可夫模型的扩展，它是在马尔可夫链的基础上发展起来的。马尔可夫链是马尔可夫随机过程的特殊情况，即马尔可夫链的状态和时间参数都是离散的马尔可夫过程。它的 $m+1$ 时刻的状态只和 m 时刻的状态有关，而与之前的状态无关。实际中，马尔可夫链的每一个状态可以对应于一个可观测到的物理事件，比如，天气预测中的雨、晴、雪等，根据这个模型可以算出各种天气在某一时刻出现的概率，如图 6-5 所示。

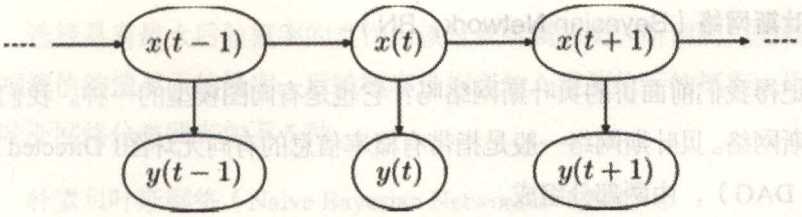


图 6-5 马尔可夫链

由于实际问题比马尔可夫链模型所描述的更为复杂，观察到的事件并不是与状态一一对应的，而是通过一组概率分布相联，这就是 HMM。HMM 是一个双重的随机过程，其中之一是马尔可夫链，描述状态转移。另外一个随机过程描述状态和观察值，它并不是与状态一一对应的，需要通过一个随机过程感知状态的存在及特性。这就是 Hidden 的由来。

一个具体的例子为 phone HMM，这是一个语音识别的例子，根据单词的发音可以建立 HMM，一个随机的有限自动机，每个状态都产生一个 evident 或 observation。识别出得到的声音特征为观察值 evident，从 start 到 end 计算得到概率最大的 HMM，最终状态即为识别到的单词，如图 6-6 所示。

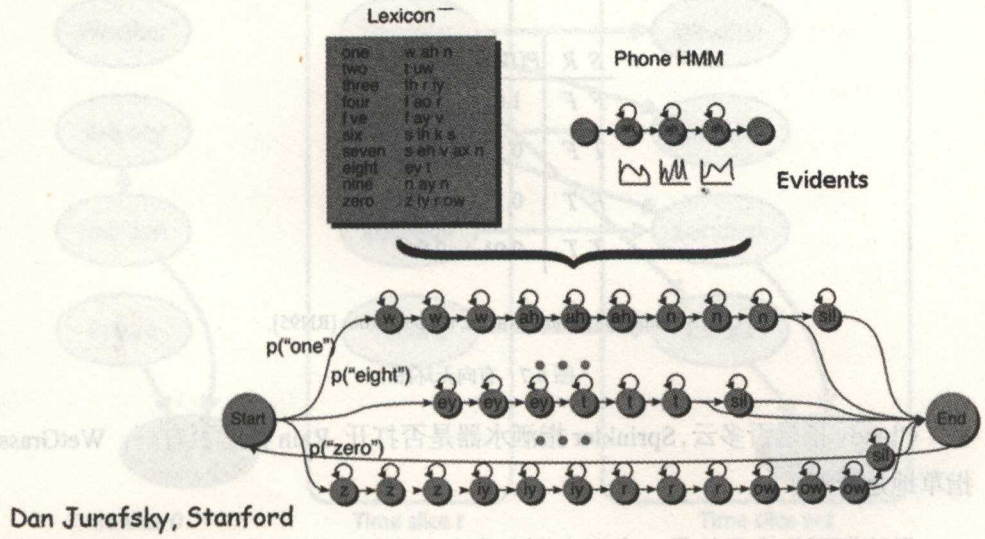
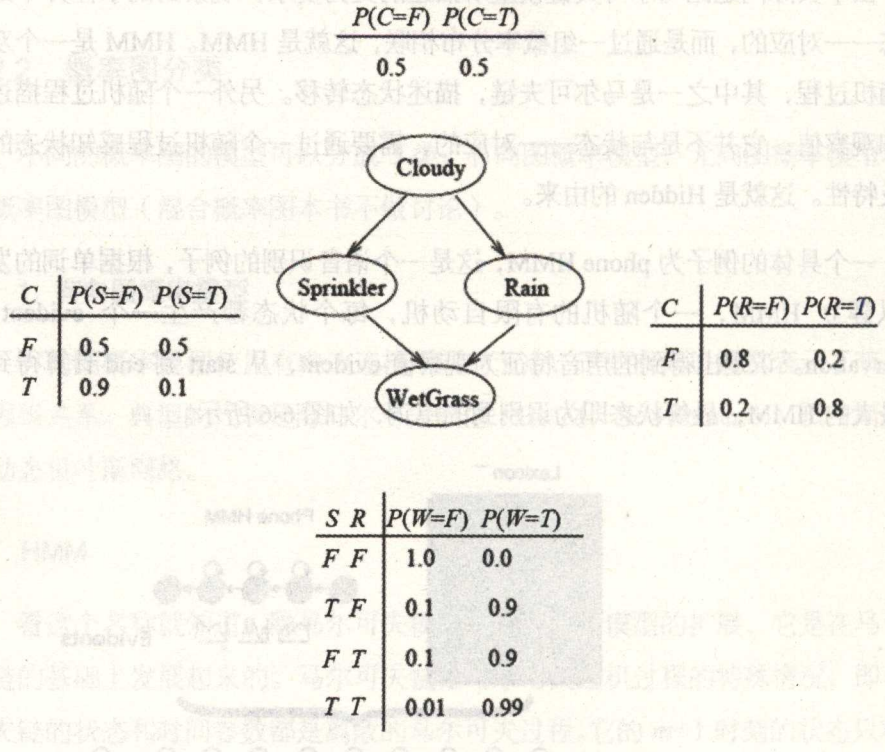


图 6-6 语音识别 HMM

贝叶斯网络（Beyesian Network，BN）

还记得我们前面讲的贝叶斯网络吗？它也是有向图模型的一种。我们再回忆下贝叶斯网络。贝叶斯网络一般是指带有概率信息的有向无环图（Directed Acyclic Graph，DAG），由两部分组成。

图6-7所示的节点表示的是随机变量，节点间的连接表示了可能的因果关系。



A simple Bayesian network, adapted from [RN95].

图 6-7　有向无环图

Cloudy 指是否多云，Sprinkler 指洒水器是否打开，Rain 指是否有雨，WetGrass 指草地是否湿了。

贝叶斯网络表现的是一个联合概率分布，它的一个用途是作为分类器，即通过某对象的先验概率，利用贝叶斯公式计算出其后验概率，即该对象属于某一类

的概率，选择具有最大后验概率的类作为该对象所属的类。所谓的先验概率是指在没有观测值的情况下的概率，后验概率是指在加入观测值后的概率。用的比较多的贝叶斯网络分类器有如下 5 种。

- 朴素贝叶斯网络（Naive Bayesian Networks，NBN）。
- 通用贝叶斯网络（General Bayesian Networks，GBN）。
- 增强型朴素贝叶斯网络（Tree-Augmented Naive Bayes，TAN）。
- 马尔可夫毯贝叶斯网络（Markov Blanket Bayesian Networks，MBBN）。
- 动态贝叶斯网络（Dynamic Bayesian Networks，DBN）。

贝叶斯网络只能反映事物的静态特征，也就是在某一个时刻事物不同特征的依赖关系。在现实生活中，存在着很多的动态随机过程，DBN 就是用来对这些过程建模的方法之一。动态表示的是一个动态的系统，而系统的结构并不随时间变化。DBN 服从马尔科夫特征：t 时刻系统的所有变量的概率分布只与 t-1 时刻的状态变量概率分布相关。HMM 可以当作 DBN 的一种特殊情况，如图 6-8 所示。

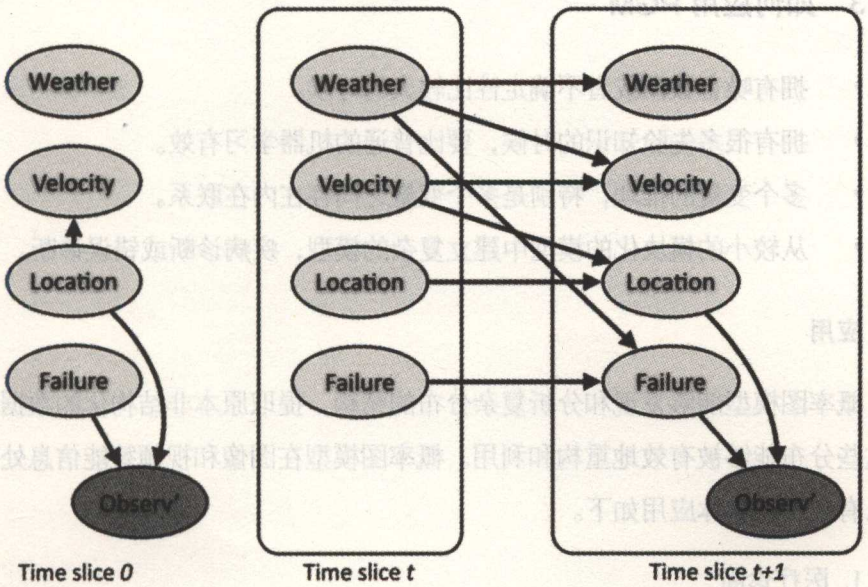


图 6-8　动态随机过程

2. 无向图概率模型

无向图概率模型用来建立随机变量间的空间相互关系或只是相互依赖性。典型的代表是马尔科夫随机场和条件随机场。无向连接通常捕捉一对结点之间的互相依赖关系。

马尔科夫随机场（Markov Random Fields，MRF）

MRF 也叫马尔科夫网（Markov Network，MN），是关于一组有马尔科夫性质随机变量 X 的全联合概率分布模型。一方面，MRF 可以表示 BN 无法表示的依赖关系，如循环关系；另一方面，它不能表示 BN 所表示的推导关系。

当概率分布为正时，被称为 Gibbs 随机场，因为根据 Hammersley–Clifford 理论（MN 与 Gibbs 分布的等价性）和局部马尔科夫性质，无向图的概率分布可以被定义为 Gibbs 公式。

6.2.3 如何应用 PGM

- 拥有噪音数据或者不确定性比较大的时候。
- 拥有很多先验知识的时候，要比普通的机器学习有效。
- 多个变量的推理，特别是多个变量之间存在内在联系。
- 从较小的模块化的模型中建立复杂的模型，疾病诊断或错误诊断。

应用

概率图模型能够发现和分析复杂分布的结构，提取原本非结构化的数据，使得这些分布能够被有效地重构和利用。概率图模型在图像和视频智能信息处理领域已有应用。具体应用如下。

1. 医疗诊断

2. 故障诊断

3. 自然语言处理

4. 语音识别

5. 社会化模型

6. 计算机视觉（图片分割、3D 重建和全面场景分析）

7 杂项

7.1 如何为不同类型的问题选择模型

DL4J 深度学习算法和应用领域如图 7-1 所示。

Data Sector	Use Case	Input	Transform	Neural Net
Text	Sentiment analysis	Word vector	Gaussian Rectified	RNTN or DBN (with moving window)
	Named-entity recognition	Word vector	Gaussian Rectified	RNTN or DBN (with moving window)
	Part-of-speech tagging	Word vector	Gaussian Rectified	RNTN or DBN (with moving window)
	Semantic-role labeling	Word vector	Gaussian Rectified	RNTN or DBN (with moving window)
Document	Topic modeling/ semantic hashing (unsupervised)	Word count probability	Can be Binary	Deep Autoencoder (wrapping a DBN or SDA)
	Document classification (supervised)	TF-IDF (or word count prob.)	Binary	Deep-belief network, Stacked Denoising Autoencoder
Image	Image recognition	Binary	Binary (visible and hidden)	Deep-belief network
		Continuous	Gaussian Rectified	Deep-belief network
	Multi-object recognition			Convolutional Net, RNTN (image vectorization forthcoming)
	Image search/ semantic hashing		Gaussian Rectified	Deep Autoencoder (wrapping a DBN)
Sound	Voice recognition		Gaussian Rectified	Recurrent Net
				Moving window for DBN or ConvNet
Time Series	Predictive analytics		Gaussian Rectified	Recurrent Net
				Moving window for DBN or ConvNet

图 7-1 DL4J 深度学习算法和应用领域

DL4J 给出了一些深度学习的算法和对应的领域，大家可以参考。对于文本领域来说，整体选择 RNTN（递归张量神经网络）或 DBN 来处理，RNTN 是 RNN（循环神经网络）的一个变种，目前笔者所从事的文本预测和对话也使用了 RNN。

对于文档（相对于文本更大）来说，利用自编码器或深信度网络能更好地解决问题。

对于图像来说，深信度网络或卷积神经网络用得比较多，对于图片搜索或图片语义来说自编码器更好。

对于声音识别，RNN 更好。

对于时间序列的，比如预测分析，DBN 或 DNN 能更好的处理问题。

7.2 我们如何学习“深度学习”

先学习什么：如本书目录安排的那样，我们需要先了解神经元→神经网络，接着了解神经网络上的各种算法，以及这些算法的意义，有些属于深度范畴，有些不属于深度范畴但也需要了解，比如 SVM、贝叶斯、K-Means 等这些都是非常经典的算法。在深度出来之前，它们都运用在各方面，直至现在仍在运用中，有些甚至还和神经网络结合在一起形成新的算法，比如自动编码器里面提到的分类器 SVM 等。

如果你英文比较好可以直接读 deeplearning 网站 reading-list 里提到的读物（http://deeplearning.net/reading-list/），当然有些比较晦涩。

作为普及性读物，本书没有深入讲解算法推导，而是主要集中在算法的使用层面讲解。幸运的是，目前来说，对于几种主流的语言都有人在给我们无偿提供深度学习的类库，包括但不限于 Java、Python、MATLAB、C++ 等，读者大可以按照自己的喜好去选择。

在现实生活中，大部分人（包括我自己）研究的是怎么使用深度学习，怎么

更好地用深度学习为生活服务，而不是推导算法，毕竟我的数学没有那么好。

对于深度学习和深度相关算法的理解，首推吴恩达的课程，他的简介链接 http://cs.stanford.edu/people/ang/，他的课程的中文翻译地址为 http://ufldl.stanford.edu/wiki/index.php/UFLDL 教程（注意，链接包括“教程”中文汉字），Andrew 还有个身份：Coursera 的创始人，对于吴恩达来说，Coursera 是他的传播工具，而深度学习是他的主要研究方向。

在整体层面，我不停地穿插各种代码，并且结合书本内容做出解释，代码的学习和运用是非常重要的，最终深度学习要产生生产力，需要大家结合应用，而这些都需要代码来实现。如果读者有代码基础，能很快理解书中提到的代码。

在完成以上学习和练习后，读者可以进阶，看看现在最前沿的研究，比如 2013 年的 Youtube 视频：Deep Learning for Vision: Tricks of the Trade - Bay Area Vision Meeting，链接为 https://www.youtube.com/watch?v=clgMTk5V2Sk。

Amazon 公司的成果 Amazon Machine Learning: use cases and a real example in Python，链接为 http://cloudacademy.com/blog/aws-machine-learning/。

了解大规模并行计算的成果怎么和深度学习结合：http://djt.qq.com/article/view/1239。

也有些非技术性的文章：IEEE 深度对话 Facebook 人工智能负责人 Yann LeCun：让深度学习摆脱束缚，链接为 http://wwww.huxiu.com/article/109035。

7.3 如何理解机器学习和深度学习的差异

Facebook 的 LeCun 给了两张图（如图 7-2 和图 7-3 所示），用图像识别举例，说得简单明了。

图 7-2 所示为传统模式进化到深度学习，差异就在特征提取上！传统模式识别的方式提取后就进行训练，提取阶段主要用些分割组合手段，最主要的是没有

层级提取的概念。

中间部分是主流模式识别，已经开始中级特征抽取，这样的特征抽取只是一级抽取，并且这个过程不用训练数据。

底部说明深度学习，两个特征，第一是分层概念（很重要），第二是经过有监督训练。

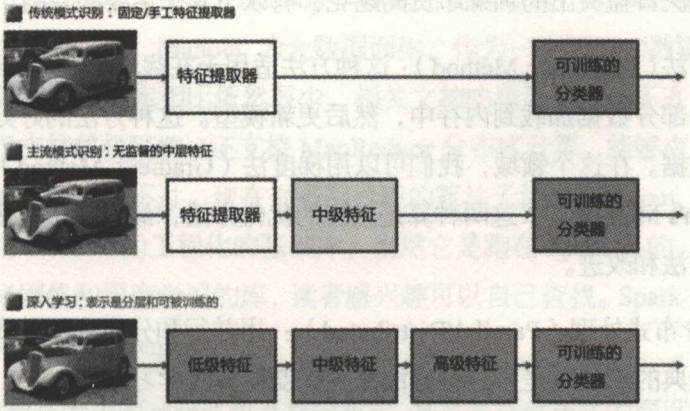


图 7-2　深度学习和传统机器学习的区别

那么，如何体现深度学习和一般神经网络的区别呢？一句话：只要有超过一个层级的非线性特征转换（抽取），我们就能定义它是深度的。

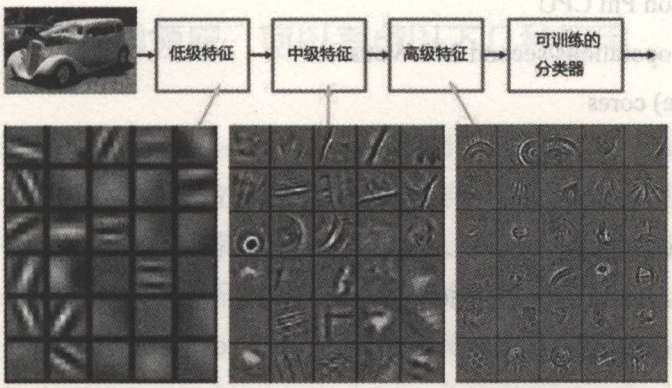


图 7-3　深度学习分层提取特征

7.4 大规模学习（Large Scale Learning）和并行计算

我们现在处在大数据时代，一个能称为大数据的范畴都得是 PB 级以上了，比如运营商的上网行为记录，每天的量是 50PB，这种规模的数据下，还要做深度学习算法的训练是非常昂贵的。

怎么解决日益突出的训练昂贵问题呢？有以下几个主要的方向。

随机方法（Stochastic Method）：这种方法适用于在线学习（online learning），每次只把一部分数据加载到内存中，然后更新模型。这种方法的好处是可以有效地处理大数据。在这个领域，我们可以用梯度法（Gradient Method）和牛顿迭代法（Newton's Method）。这两种算是主流的优化方法，最近几年很多论文都是在讨论这些方法和改进。

并行/分布式处理（Parallel/Distributed）：用并行和分布式的方式提高学习速度。很多经典的例子就是用这种方法提高深度模型的学习速度。

并行处理的类协处理器，目前正在使用的有 Intel 公司和 NVIDA 公司的产品。它们都是利用类协处理器原理加速某些特定的矩阵运算。

下面是两家公司的比较有代表性的产品和规格。

Intel Xeon Phi CPU
2x1012 operations/second 240 Watts
60 (large) cores
$3000

NVIDIA Titan-Z GPU
8x1012 operations/second 500 Watts
5760 (small) cores
$3000

看起来已经很强大了，但这与我们大脑的计算能力还相差 100 万倍。根据摩尔定律，我们需要至少 30 年才能拥有此计算能力，也就是说，至少到 2045 年，我们的单卡能力才能达到人脑能力。

在此之前，我们能通过分布式处理，并联多块板卡或多台计算机加速。下面，我们讲一些可用的分布式框架。

搭建分布式机器学习平台：Hadoop 当然是目前的主流，Mahout 是 Apache 旗下专门的分布式算法库，Mahout 为大数据而生，作为一个新生的数据挖掘工具，它所支持的算法与其他库相比依然很少，相关文档的质量也良莠不齐，但是它的优势在于不仅支持单机环境，还支持 MapReduce 分布式计算，能够应对一般算法库无法处理的大数据。不过，现在很多刚出现的算法无法被 Mahout 支持，现在我们一般使用 Mllib 作为工程化的基础库，当然它是跑在 Spark 上的，基于 Spark 还有很多神经网络和深度学习的库，读者感兴趣可以自己查找。Spark 还有个优良特性，它支持 Python client。

当然，我现在所用的 mxnet 也支持分布式，甚至支持分布式的深度学习方案，甚至在前不久，谷歌的 tensflow（深度学习库）也发布了分布式版本。以后，深度学习必然跟分布式联系紧密，计算能力也变得越来越重要。在这个领域，还有很多需要持续优化和研究的地方。

7.5 如果喜欢应用领域，可以考虑以下几种应用

推荐系统（Recommender System）：这算是机器学习领域里最为成功的应用，但还是有很多东西值得深入研究。推荐系统最重要的问题是冷启动问题，就是去解决对新用户，新商品如何推荐的问题，以及怎么把多种异类信息有效地结合在模型本身上一直是个有待持续研究的问题。在推荐系统问题上，最常见的方法是矩阵分解（Matrix Factorization），它的各种提高版本出现在 2015 年的很多论文上。

文本分析和挖掘：很多经典的问题，比如情感挖掘、文档相似性、情报收集等，正好笔者做这方面的研究，现在已经搭建起面向开发者的公有云平台，除了提供上述文本分析外，还有对话机器人Demo，有兴趣可以尝试（http://acnlp.com）。

广告投放：谷歌、百度等公司大部分的收入来自于广告。核心问题就是去解决：合理地选择广告，提高用户点击率。

社交网络分析：比如垃圾邮件处理（垃圾信息处理）、地理位置信息分析、社会影响分析、因果分析系统、链接预测、使用社会化信息推荐、社会动态的评价等。

金融上的应用：预测价格，预测金融事件，谷歌趋势分析等。笔者正在编写上证个股指数预测模型并基于社交化网络分析、挖掘人们的情绪观点，以便更准确地预测股票。

电子交易上的应用：反欺诈交易等。

系统上的应用：能耗分析、系统 BUG 的检测（典型的例子有吴恩达领导的百度磁盘正常度预测，提前预测磁盘工作状态并进行替换，以提高整个系统的稳定性）。

电力系统：智能电网和智能电表，利用每户的用电信息，反应整个城市的各个指标，给政府等相关机构提供重要的决策参考。

除此之外，机器学习在航空航天（比如 NASA）、军事等领域都有一些身影。

7.6 类脑：人工智能的终极目标

我们说了这么多深度学习的模型概念，总的说来是为了解决现实世界中的一些问题。所有的模型都有这么一个共性，就是如何学习现实社会中的知识（训练），如何解决问题。深度学习的各种模型其实可以模拟人脑学习的基本模式。那问题来了，从神经网络到深度学习只是更进一步地提炼了数理模型模仿度。我们是否

可以更进一步，完整地模拟神经元及脑中各个功能区？如果还原了一个完整的大脑，那么所有功能会有真正的人工智能诞生吗？如果我们知道了如何模拟一个大脑，那么我们能否模拟一个更强的脑，这个大脑是不是可以链接到人类世界所有的知识库去学习，并诞生几乎无限的创造力？这就是类脑的作用。

说到模拟大脑，从目前的技术手段来看，有很多种方法。最现实的还是用硅基 CPU 去模拟，由于 CPU 的运作机理和大脑生物运作机理有很多不同，所以只能通过数学模型的方式去模拟，只不过这个数学模型非常复杂，造成了模拟一部分功能都需要消耗较多的处理能力。

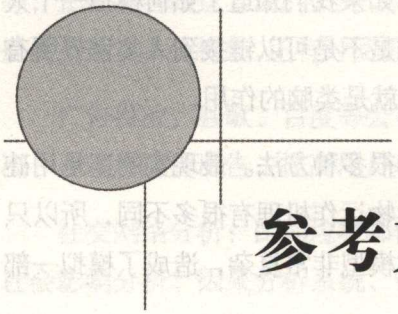
对于这个问题，我国和世界研究领域的先进水平基本保持一致，有两个解决问题的思路。

（1）开发一种新型的 CPU，也许名字都不叫 CPU 了，但不管叫什么，它只用干一件事情——非常有效率地模拟神经元。这种 CPU 将极大地提高生产效率，用极小的代价模拟神经元。中科院的神经网络芯片和 IBM 刚推出的芯片都属于这种范畴。

（2）完全抛弃 CPU，也不用数理模型模拟了，直接用光电器件去模拟神经元。也就是利用另一种信号搭建神经网络结构。不问为什么，只是利用光电系统对大脑结构进行复制。

目前，这个领域国内由北京大学计算机系主导研究。这就是“第三类智能”的概念，也就是生命的基础可能不是基于 CPU 模拟（硅基），也不是基于地球上的蛋白质本身（碳基），可能由第三种或者更多种的方式实现。

最后借用类脑计算研究中心学术委员会主任张钹院士的话：类脑计算比曼哈顿计划更有挑战性。



参考文献

[1] 成素梅 郝中华.《BP 神经网络的哲学思考》.2008 年，《科学技术与辩证法》，第 4 期.

[2] 张建军 王士同 王骏.迁移学习数据分类中的 ESVM 算法.2012 年 4 月，《计算机工程》第 38 卷第 8 期.

[3] 维基百科 https://www.wikipedia.org/.

[4] Yoshua Bengio《LearningDeep Architectures for AI》 Now Publishers Inc October 28, 2009.

[5] A Deep Learning Tutorial: From Perceptrons to Deep Networks http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks.

[6] UFLDL-斯坦福大学 Andrew Ng Deep Learning 教程，http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial.

[7] 深度学习教程系列，http://blog.csdn.net/zouxy09/article/details/14222605.

[8] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks Advances in neural information processing systems. 2012: 1097-1105.

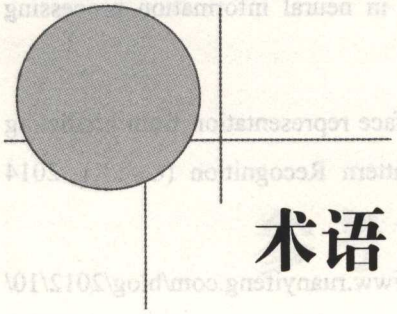
[9] Sun Y, Wang X, Tang X. Deep learning face representation from predicting 10,000 classes Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on June 2014.

[10]贝叶斯推断及其互联网应用 http://www.ruanyifeng.com/blog/2012/10/spelling_corrector.html.

[11]深度学习概述：从感知机到深度网络 http://www.cnblogs.com/huicpc0212/p/4761875.html.

[12]卷积神经网络 http://blog.csdn.net/stdcoutzyx/article/details/41596663.

[13]AlphaGo 在人机大战中风光了一把，但它的弱点依然存在 http://www.leiphone.com/news/201603/VARZ2sn7aC2DPBkw.html.



术语

1. 图灵：全名艾伦·麦席森·图灵，英国人，因性倾向遭到当时的英国政府迫害，职业生涯尽毁。他可以说是人工智能之父，笔者十分佩服其才智。
2. 决策树：是一个预测模型；它代表的是对象属性与对象值之间的一种映射关系。树中每个节点表示某个对象，而每个分叉路径则代表某个可能的属性值，每个叶节点对应从根节点到该叶节点所经历的路径所表示的对象的值。
3. 条件概率：就是事件 A 在另外一个事件 B 已经发生的条件下的发生概率。条件概率表示为 $P(A|B)$，读作“在 B 条件下 A 的概率”。
4. 树突、轴突：神经元的输入和输出部分。
5. AND/XOR/OR：数学逻辑运算。
6. 人工神经元：是一种模仿生物神经元的结构和功能的数学模型或计算模型。
7. 感知机：它被视为是一种最简单的前馈神经网络，是一种二元线性分类器。

8. 前馈神经网络：最简单的人工神经网络类型。在它的内部，参数从输入层向输出层单向传播。

9. 特征：本书中指将现实生活中的事物的部分特点提取并抽象出一种数学或物理模型。

10. 特征粒度：提取特征的维度。

11. 浅层学习/深度学习：相对概念，深度学习相对浅层学习抽象层级要多。

12. BP 算法（反向传播算法）：是一种监督学习算法，常被用来训练多层感知机，利用反向传播原理修正权值。

13. 自动编码器（AE）：自动编码器就是一个运用了反向传播进行无监督学习的神经网络，学习的目的是为了让输出值和输入值相等。

14. RBM（限制波兹曼机）：是一种可通过输入数据集学习概率分布的随机生成神经网络。

15. 概率模型：是用来描述不同随机变量之间关系的数学模型，通常情况下刻画了一个或多个随机变量之间的相互非确定性的概率关系。

16. 能量模型（EBM）：基于能量的模型，把我们关心的变量的各种组合和一个标量能量联系在一起。我们训练模型的过程就是不断改变标量能量的过程。

17. DBN（深度信度网络）：通过自底向上组合多个 RBM 可以构建一个 DBN，利用非监督贪心逐层训练算法，解决深层结构相关的优化问题。

18. CNN（卷积神经网络/ConvNets）：是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。

19. 概率图：是一种使用图来表达随机变量之间条件独立性的概率模型。

20. 贝叶斯定理：事件 A 在事件 B（发生）的条件下的概率。

21. SVM（支持向量机）：监督学习方法，属于一般化线性分类器。这种分类器的特点是它们能够同时最小化经验误差与最大化几何边缘区，因此支持向量机也被称为最大边缘区分类器。

22. K-Means：把 n 个点（可以是样本的一次观察或一个实例）划分到 k 个聚类中，使得每个点都属于离它最近的均值（此即聚类中心）对应的聚类。

23. Java：一种面向对象的高级语言。

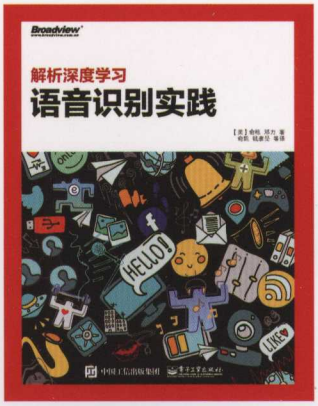
24. Python：是一种解释型的计算机程序语言，具有近 20 年的发展历史。它包含了一组功能完备的标准库，能够轻松完成很多常见的任务。

25. MATLAB：是一款由美国 TheMathWorks 公司出品的商业数学软件。MATLAB 是一种用于算法开发、数据可视化、数据分析及数值计算的高级技术计算语言和交互式环境。

26. C++：是一种广泛使用的计算机程序设计语言。

27. 并行计算：一般是指许多指令得以同时进行的计算模式。

28. NASA：国家航空航天局（英语：National Aeronautics and Space Administration，缩写为 NASA），是美国联邦政府的一个行政机构，负责制定、实施美国的民用太空计划，并开展航空科学暨太空科学的研究。



【美】俞栋 邓力 著
俞凯 钱彦旻 等译

欢迎投稿

邮箱：Ljiao@phei.com.cn
电话：010-88254395
新浪微博：@皎丫子

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

神经网络与深度学习

未来，机器AI有可能替代人类的工作吗？重复性质的工作首先会被替代，而一些表达人类情感和艺术类的工作呢？我认为，AI能产生比人类更有想象力或更有创造力的作品，某种意义上说，甚至比人类的作品更好！它们的差别在于，人类作品能和人类自己保持更深层次的情感共鸣，或者摸到人类更深层的脾气，更容易贴近人类的某一类情感需求。这会导致艺术家的工作内容发生转变，虽然还叫艺术家，但已经不独立作画或写作了，更有可能是借助AI完成作品。

另外，人类的直觉很重要。无论是工作还是生活，人类靠直觉处理很多事情，而AI的直觉有可能依靠大规模训练和结构的进化在某一天超过人类。到那时，人类可能无法理解机器的直觉或思维，人类真的创造了另一个“物种”！

博文视点Broadview

新浪微博 weibo.com
@博文视点Broadview

策划编辑：刘　皎
责任编辑：郑柳洁
封面设计：吴海燕

上架建议：人工智能

ISBN 978-7-121-28869-2

定价：59.00元